Adaptive Dynamic Software Diversity: Towards Feedback-Based Resilience

Andrea Höller, Tobias Rauter, Johannes Iber, and Christian Kreiner
Institute for Technical Informatics, Graz University of Technology, Austria
{andrea.hoeller, tobias.rauter, johannes.iber, christian.kreiner}@tugraz.at

***Abstract*—Embedded systems are increasingly based on commercial off-the-shelf processors that limit the use of hardware-based reliability techniques. At the same time faults are on the rise due to shrinking hardware feature sizes, increasing software complexity, and security vulnerabilities. Since such faults cannot be completely prevented, systems have to cope with their effects. However, there is a lack of methods that allow embedded systems to recover from detected faulty states. In order to contribute towards filling this gap, we introduce the concept of adaptive dynamic software diversity. The main idea is to create a feedback-based system that adapts the execution of the program in such a way that a fault is bypassed regardless of its root cause. To achieve this, we propose the use of automated diversity techniques.**

I. Introduction

Embedded systems with high reliability and availability requirements have to realize ever more features and have increasing demands on computing performance. Consequently, designers often use commercial off-the-shelf (COTS) processors that offer cost efficiency and high performance. At the same time, embedded systems have to cope ever often with unforeseen scenarios caused by an increasing number of faults that can have numerous causes:

- *Random hardware faults*, such as soft errors and permanent errors due to manufacturing, process variations, ageing, etc. occur more and more frequently due to continuing structure downscaling.
- *Software faults* are increasing due to growing software complexity.
- *Security attacks* pose an emerging risk, since ever more systems are interconnected.

Faults cannot be prevented, so they remain in every complex system. Consequently, to make systems resilient, they have to cope with changing circumstances regardless of their root cause. In order to deal with unforeseen events the idea of software self-adaptability has received attention [1]. For example, self-healing systems autonomously detect and recover from faulty states by changing their configuration. However, so far these techniques are mainly used in complex server systems.

Methods for embedded systems to recover from an unhealthy state are still a research challenge. Although hardware faults can be bypassed with self-modifying hardware (e.g. [2]), this technique is not applicable for COTS hardware and only offers limited flexibility. Thus, there remains the need for sophisticated software-based methods to handle unforeseen scenarios caused by faults. In this paper, we propose adaptive dynamic software diversity as means of adapting the system in order to mask faults. We introduce this method as a high-level concept that is based on automatic diversity approaches.

II. Automated Diversity

Recently, automated diversity gained attention in the security domain as a technique of diversifying each deployed program version [3]. This forces attackers to target each system individually. However, it can also increase the hardware and software fault tolerance by automatically introducing diversity in redundant systems [4], [5].

In contrast to static techniques (e.g. diverse compiling) that are applied before deployment, dynamic software diversity techniques integrate randomization points in the executable. Then, the same program can perform diverse executions leading to the same results [6]. Diversity in execution can mean, for example, diverse performances, diverse memory locations, or diverse order of execution. Table I shows examples of dynamic diversity techniques. Data re-expression is a well-established method of obtaining data diversity by transforming the original input to produce new inputs to redundant variants [7]. After execution the distortion introduced by the re-expression is removed before comparison. So a given initial data within the program failure region can be re-expressed to an input data that circumvents the faulty region [8]. Dynamic diversity techniques can be efficient to in addressing memory-related faults as shown in [9], [10], [11].

TABLE I. Examples of Dynamic Software Diversity Techniques and their Adjustable Parameters

Randomization method	*Parameter*
Memory gaps between objects [11]	Gap size
Changing base address of program [11]	Base address
Changing base address of libraries and stack [9]	Base address
Permutation of the order of routine calls variables [11]	Order of calls
Permutation of the order of variables [10]	Order of variables
Insertion of NOP instructions [12]	Number of NOPs
Data re-expression / data diversity ($in=f(in,k)$, $out = f^{-1}(out,k)$) [7]	Parameter in re-expression algorithm (k)

III. Adaptive Dynamic Software Diversity

A promising approach for resilience, is software that offers a reliable operation despite uncertain environments. Hardware faults, software bugs, or security exploits can be regarded as sources of uncertainty in the operation that has to be handled.

For example, permanent hardware faults cannot be fixed during runtime. Thus, the software has to change the way it uses the faulty hardware such that the fault is masked. Adapting the software execution is probabilistic and does not require knowledge about the exact root cause of the fault. We propose to learn from detected anomalies and to adapt the software by diversifying the execution with adaptive dynamic software diversity (ADSD). We define ADSD as

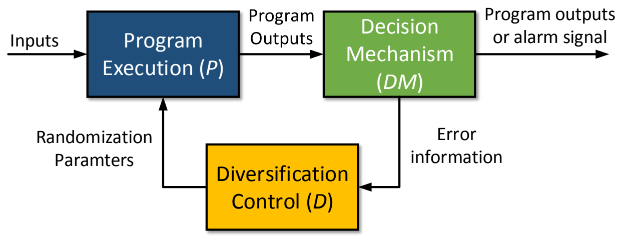

Fig. 1. Basic structure of ADSD. Based on information of a monitoring component (DM), a diversification controller decides whether and how to reconfigure the randomization mechanism of the main program.

a method to automatically and dynamically diversify the way of execution (e.g, used resources, executed code) in such a way that it learns from previous observed anomalies in order to increase the fault tolerance regardless of the fault's cause.

A. Basic Structure

Fig. 1 shows the basic structure of an ADSD system. Typically, a fault tolerant system contains the program, which performs the intended functionality of the system and a decision mechanism (DM) that monitors the program execution [8]. The DM detects anomalies, indicates alarms and decides which outputs to forward. For example, the diagnosis could be a plausibility check, a voter of a redundant system, or a self-aware technique that detects anomalies. Additionally, we propose to add a component denoted as diversification control that creates a feedback-loop. This component manages the ADSD by collecting and analyzing data on detected anomalies obtained from the DM. We propose to design the program in such a way that it can be randomized during execution according to parameters that can be adjusted during runtime (see Table I). Then, the diversification control can decide to alter the execution by changing one or multiple randomization parameters. Finally, the program reconfigures itself by using the adapted parameters.

B. ADSD as an Autonomic Control Loop

Feedback loops are essential for self-adaptive systems [1]. Theories about feedback-based systems from control engineering and nature may supply adaptive software techniques. As shown in Fig. 2 the generic model of a feedback loop involves four key activities: collect, analyze, decide and act [13]. In classical self-adaptive systems sensors and probes collect data from the executing system and its current state. This data is then filtered, preprocessed and collected. A diagnosis mechanism then analyzes the data by recognizing trends and identifying symptoms. Then, it tries to predict the future in order to decide how to act so that the reliability is increased.

ASDS techniques can be represented as such an autonomic control loop (see Fig. 2). The DM *collects* data about detected anomalies. This data is forwarded to the diversification control, which *analyzes* the trend of the anomalies. Furthermore, the diversification control keeps track of the previously used randomization parameters. If this trend indicates that a specific component of the system is faulty, then the system has to learn from this observation and it *decides* to change its behavior. Therefore it *acts* by adapting the parameters under consideration of previously changed parameters and their effects.

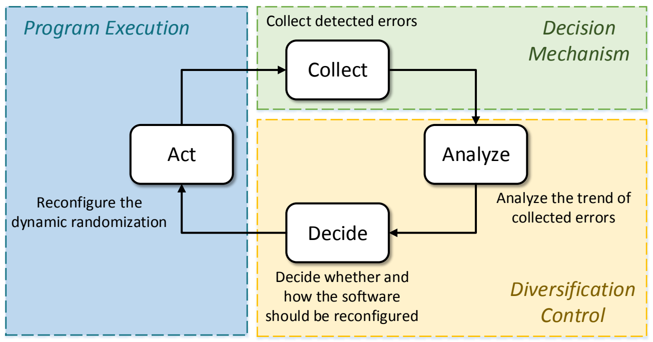
Program Execution
Collect detected errors
Decision Mechanism
Collect
Act
Analyze
Analyze the trend of collected errors
Reconfigure the dynamic randomization
Decide
Decide whether and how the software should be reconfigured
Diversification Control

Fig. 2. ADSD as an autonomic control loop. The decision mechanism collects error statistics. The diversification control analyzes trends of these statistics and decides whether the software should be reconfigured. If this is the case, the program acts by reconfiguring itself. Adapted from [13]

IV. Conclusion

Here, we presented the idea of increasing the resilience of COTS-based systems with ADSD. However, the actual realization of ADSD depends on the application and the considered fault model. Thus, we plan to develop ADSD techniques for specific complex applications. Furthermore, we hope to encourage further researchers to explore techniques based on the promising yet challenging idea of ADSD.

References

[1] Y. Brun, G. D. M. Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezze, and M. Shaw, "Engineering Self-Adaptive Systems Through Feedback Loops," *Software Engineering For Self-Adaptive Systems*, 2009.

[2] S. M. A. H. Jafri, S. J. Piestrak, O. Sentieys, and S. Pillement, "Design of a Fault-Tolerant Coarse-Grained Reconfigurable Architecture : A Case Study," *International Symbosium on Quality Electronic Design*, 2010.

[3] P. Larsen, A. Homescu, S. Brunthaler, and M. Franz, "SoK: Automated Software Diversity," in *Proceedings of IEEE Security & Privacy*, 2014.

[4] G. Gaiswinkler and A. Gerstinger, "Automated software diversity for hardware fault detection," in *Emerging Technologies & Factory Automation*, 2009.

[5] A. Hoeller, N. Kajtazovic, T. Rauter, K. Roemer, and C. Kreiner, "Evaluation of Diverse Compiling for Software Fault Detection," *Design, Automation and Test in Europe*, 2015.

[6] B. Baudry and M. Monperrus, "The Multiple Facets of Software Diversity : A Survey," 2014.

[7] P. Ammann and J. C. Knight, "Data Diversity: An Approach to Software Fault Tolerance," *IEEE Transactions on Computers*, 1988.

[8] L. Pullum, *Software Fault Tolerance: Techniques and Implementation*, 2001.

[9] M. Chew and D. Song, "Mitigating Buffer Overflows by Operating System Randomization," Tech. Rep., 2002.

[10] S. Bhatkar, D. DuVarney, and R. Sekar, "Address Obfuscation: An Efficient Approach to Combat a Board Range of Memory Error Exploits," in *USENIX Security Symbosium*, 2005.

[11] S. Bhatkar, R. Sekar, and D. DuVarney, "Efficient Techniques for Comprehensive Protection from Memory Error Exploits," in *USENIX Security Symbosium*, 2005.

[12] A. Homescu, S. Neisius, P. Larsen, S. Brunthaler, and M. Franz, "Profile-guided automated software diversity," *IEEE/ACM International Symposium on Code Generation and Optimization*, 2013.

[13] S. Dobson, S. Denazis, A. Fernandez, D. Gaiti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli, "A Survey of Autonomic Communications," *ACM Transactions on Autonomous and Adaptive Systems*, 2006.