Iterative Dependable Systems Engineering with *Fail**

Horst Schirmeier, Technische Universität Dortmund,
Embedded System Software Group

Recent studies suggest that future microprocessors need low-cost fault-tolerance solutions for reliable operation. The DFG rates this problem so important that they are funding a Priority Program – SPP 1500 "Dependable Embedded Systems" – to investigate solutions and tradeoffs on both software and hardware layers.

On the software side, several competing error detection and correction methods have been shown to increase the overall resiliency when applied to *critical spots* in the system. Some projects harden the OS kernel against soft errors, e.g., our DanceOS project uses Aspect-Oriented Programming and *Generic Object Protection* in *eCos*. Others extend the OS by dependability services, e.g., Döbel's *Romain* adding transparent replication to Fiasco.OC, or Heinig's *FAME* microvisor collecting hardware-detected errors and dispatching them to application-specific handlers. The authors of these works share two common problems:

- How can we identify the *critical spots* in our system that are particularly susceptible to certain types of hardware faults? Hence, which parts of our system should get a high priority for protection measures, as their resiliency has the largest impact on the whole system?
- How can we systematically *test the functionality* of our fault-tolerance measures, and how can we *quantify their effectiveness*? Does our fault-tolerance measure increase the overall fault resiliency, or does the increased contact surface (in terms of used memory, and executed instructions) diminish the gains?

In this talk I will present our fault-injection and dependability-analysis framework Fail* that aims at solving both problems: On the one hand, the tool allows to identify particularly vulnerable code sections, important variables, data types, or program phases. On the other hand, it can be used for systematic testing, and to quantify and compare the effectiveness of fault-tolerance measures.

Compared to the competition in the field, Fail* features excellent *scalability* (allowing to analyze large software stacks, such as complex OS setups) by massive parallelization and smart experiment-eliding techniques that minimize result inaccuracies. Additionally, the tool offers a strong *flexibility* regarding experiment setups, fault models, and target systems: Fail* can be used to analyze workloads running on different architectures and in several different simulator (e.g., Bochs/i386, gem5/ARM) and hardware (e.g., PandaBoard/ARM, Lauterbach/ARM) backends, assuming single- or multi-bit faults in, e.g., RAM or the register file. I will accompany the talk with concrete analysis examples, i.a., from a protected variant of *eCos*, and from an AN-encoded voter providing the core of a triple-modular-redundantly executed quadrocopter control application.