Formal Modeling of Safety Requirements in the Model-Driven Development of Safety Critical Embedded Systems

Dominik Sojer, Alois Knoll
Department of Informatics
Technische Universität München
85748 Garching bei München, Germany
{*sojer,knoll*}*@in.tum.de*

Christian Buckl
Cyber-Physical Systems
fortiss GmbH
80805 München, Germany
buckl@fortiss.org

***Abstract*—Safety requirements are a very important artifact in the development of safety critical embedded systems. They are usually identified during safety analyses and are used by experts as a basis for the correct selection and implementation of safety mechanisms. Various safety analysis research groups have worked on formal modeling of safety requirements with the goal of determining if a system can meet these requirements. In this abstract, we propose the application of formal models of safety requirements throughout all development phases of a model-driven development process. The safety requirements identified during safety analysis can be used to automatically generate appropriate mechanisms in the code generation phase and to verify the suitability of this mechanisms in the verification phase. By establishing safety requirements as a formal basis of all process phases, a consistent development process can be achieved.**

***Keywords*-model-driven development; safety analysis; embedded systems; safety critical embedded systems**

I. Introduction

During the development of safety critical embedded systems a very important step is the identification of hazards and their analysis, which is called safety analysis. In this phase, safety requirements are identified that have to be met by the system under development for being "safe". A safe system is defined as being free of unacceptable risks. During a conventional safety analysis, non-formal safety requirements are identified and experts use them to decide on mechanisms that should enhance the overall system safety. However, during the last years, several research groups tried to find formal ways of describing safety requirements with the goal of formal verification [1].
In this abstract, we propose to go one step further by using formally modeled safety requirements for system development in model-driven software development. The information embedded in formally modeled safety requirements can be used to select and generate appropriate mechanisms and to verify the resulting system.

II. Approach

Our approach is an extension of conventional safety analysis processes and is performed in two steps: identification of safety requirements at component level and system composition including selection of appropriate fault-tolerance mechanisms. This two steps are explained in the following.

A. Description of Components and Safety Requirements

Systems and their requirements can be described on the system level or on a more granular component level. We propose that safety requirements are so application dependent that it is not useful to define a description methodology for them on the system level. A description language for this use case would have to be nearly as powerful as a natural language because it would have to capture specifics from a very broad spectrum of application areas, like "the pressure pot is safe as long as the pressure sensor delivers correct measurements at least every 5 seconds". Therefore we propose to use conventional safety analysis techniques like fault tree analysis to refine the safety requirements to a formally defined logical component level, like the actor level of actor based systems.
Following the component based development approach of actor based systems, safety requirements should only be describable for inputs and outputs, to permit their composability. Moreover, for future analysis, it should be describable which safety requirements an actor can provide as a black box. This will be called "safety assurance" in the following to prevent confusion. However, syntactically and semantically, safety requirements are equivalent to safety assurances.
This concept and the relationship of safety requirements and safety assurances is visualized in fig. 1.
On the actor level, safety requirements can be described by an extension of McDermid's fault classes [2]. However, these fault classes are too abstract for an application in model-driven development, therefore we propose an extension to:

- wrong value with a concrete threshold
- wrong timing with a concrete threshold
- commission and omission
- wrong values of one variable in subsequent time steps
- wrong values of multiple variables at the same time

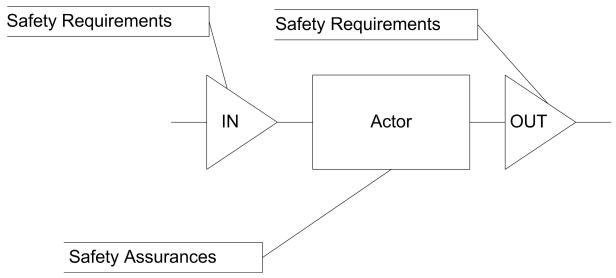

Figure 1. Safety Requirements and Safety Assurances

Wrong values and wrong timing have to be extended with concrete thresholds so that it is possible to determine the correctness of a variable in a concrete implementation. The observation of "multi-errors" (multiple wrong values of one variable in subsequent time steps and wrong values of multiple variables at the same time) is required because safety related events often possess a very specific chronological sequence and are often not limited to a single source of error.
With the help of these fault classes, safety requirements for inputs and outputs and safety assurances for actors can be described.

B. System Composition

The composability of the approach can be handled in a way that the input safety requirements of an actor can be directly linked to the output safety requirements of its predecessor. If the actor's input safety requirements are stricter that its predecessor's output safety reqirements, then the predecessor's output requirements can automatically be overwritten.
Moreover, the output safety requirements of a task can be opposed to the safety assurances of the actor to determine flaws in the actor's safety assurances. The workflow of this analysis will be described in the following.
The safety assurances of an actor can differ from the safety requirements of one of its subsequent outputs in two different ways: first, the actor may assure "safer" assurances (e.g. the actor has a lower threshold for wrong values). In this case, no additional measures have to be taken. Second, the actor may assure "unsafer" assurances than its subsequent output demands them. In this case, the actor has to take measures so that it is able to fulfill the possessed requirements.
The first step is to analyze the actor to gather information about the hardware where it is executed. This is important because only hardware faults can result in sporadic system failures, whereas software faults are, according to generally accepted theorems [3], systematic faults that cannot be handled via standard fault detection and fault handling mechanisms.
In the second step, fault detection mechanisms have to be generated that are able to detect hardware faults that result in a violation of the subsequent output's safety requirements. This approach is well aligned with up-to-date safety standards like IEC 61508, which also try to handle hardware faults by suggesting adequate fault detection mechanisms for every hardware component of a computer system.
To achieve this, the fault detection mechanisms have to have a formal description, consisting of

- affected hardware component types
- detectable fault classes
- resource consumption (e.g. execution time, memory consumption, energy requirements,...)

The affected hardware component types and detectable fault classes are necessary to select appropriate fault detection mechanisms for a given safety requirement and a given actor. The description of consumed resources is not necessary to select an adequate mechanism on the logical level, but it is very important when such a system is realized. In this case, this information can be used to not only simply select an adequate fault detection mechanism, but also to select one that does not harm the system's non-functional requirements, like deadlines or memory limitations.

III. Conclusion and Future Work

We developed the approach, presented in this paper, and will implement it in the future. The main problem that we expect is that our approach generates a huge set of safety-related mechanisms, which is probably highly redundant. With the help of the mechanisms' formal descriptions the set of generated mechanisms can be optimized, for example by minimizing the accumulated runtime of all mechanisms. However, these optimization strategies will only be a future step of our work. There is obviously no perfect optimization strategy, because the individual parts of the descriptions cannot be balanced against each other in general. For example, some systems depend heavily on runtime whereas others depend mainly on energy consumption. Therefore we will define an optimization strategy that is highly customizable so that it can be adjusted to as much application areas as possible.

References

[1] A. Joshi and M. P. E. Heimdahl, *Behavioral Fault Modeling for Model-based Safety Analysis*, Proceedings of the 10th IEEE High Assurance Systems Engineering Symposium, 2007.

[2] J. A. McDermid and D. J. Pumfrey, *A Development of Hazard Analysis to Aid Software Design*, Proceedings of the Ninth Annual Conference on Computer Assurance, 1994.

[3] T. Anderson et al., *Software Fault Tolerance: An Evaluation*, IEEE Transactions on Software Engineering, vol. 11, no. 12, pp. 1502-1510, 1985.