A Model-Based Code Generator in the Context of Safety-Critical Systems

Christian Buckl, Matthias Regensburger, Alois Knoll, Gerhard Schrott
Department of Informatics
Technische Universität München
Garching b. München, Germany
{buckl,regensbu,knoll,schrott}@in.tum.de

Abstract

Most of the existing model-based development tools focus on the pure application functionality. Non-functional features like the automatic generation of fault-tolerance mechanisms are not covered. One main reason is the inadequacy of the models being used. In addition, the code generator must be extendible to cope with the huge heterogeneity of safety-critical systems. In this abstract[1], we propose a template-based code generator and a meta-model using the concept of logical execution times as execution semantics. We will the main concepts, the possibilites and limitations of this approach.

1 Introduction

Model-based design has become state of the art in software engineering. The possibility to generate code automatically from the model accelerates the development process and is therefore very attractive. For this, several tools like Matlab/Simulink or SCADE are available. These tools focus predominantly on the functional aspects of the applications like the control functions. However, major parts especially of fault-tolerant embedded systems are related to system aspects. We understand by the term *system aspects* all non-functional aspects related to the distribution of the embedded system and the need for fault-tolerance: process management, scheduling, inter-process communication, communication within the distributed system and the fault-tolerance mechanisms. In general, these aspects are not addressed by existing tools and have to be implemented manually.

Two main reasons can be identified why tools do not cover code generation for system aspects: the absence of adequate models and the platform dependency of system level code.

Model: Concerning the models, the widely used Unified Modeling Language UML, for example, lacks the precision and rigor needed for code generation. Only a few models, such as class or state machine diagrams, can be used for automatic code generation. Especially in the context of safety-critical systems, several requirements like composability of diverse components, determinism in execution and error containment are posed on the model. A solution to this problem is the usage of domain specific languages (DSL).

Since several aspects of the whole system must be described, we decided to split up the model into five distinct sub models, each describing one aspect of the system [2]. The safety model specifies the safety functions of the system and the relevant certification guideline. Within a hardware model, the topology of the distributed system and network can be described. The software model is used to model the system architecture (software processes, communication points, interaction with the environment). We use the concept of logical execution times[3] to guarantee replica determinsm and separate the state of the system from the system behavior, thus allowing the automatic realization of state synchronization. The faults that should be tolerated by the system, as well as the behavior of affected components is specified within a fault model. Finally, a fault-tolerance model is used to specify the

[1]This work is funded by the German Federal Ministry of Education and Research BMBF under grant 01ISF12A

tests for error detection and the fault-tolerance mechanisms like triple-modular redundancy (TMR) or hot-standby.

Code Generation Architecture: The second problem is the platform[2] dependency of system level code. Since safety-critical real-time software is typically embedded in a larger system, a huge heterogeneity of the used platforms exists. Due to this variety, it is not possible to design a code generator for system aspects that supports a priori all these platforms. Rather, the code generator must allow easy extension both of the underlying model and of the code generation ability, even by the user. Template-based code generators can be used to achieve this extensibility regarding the code generation ability.

The actual functionality, e.g. the control functions, must be implemented by the application developer. Here, several tools like Matlab/Simulink are available. Figure 1 depicts the phases of a typical development process and marks the process steps where our tool is deployed.

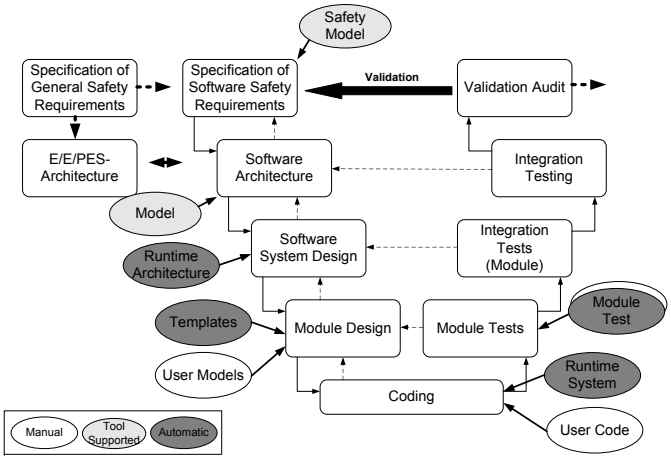

Figure 1. Tool in the Context of the V-Model

Possibilities and Limitations: Classic fault-tolerance mechanisms can be implemented in an application independent manner and therefore, also generated automatically. Examples are hot-standby or TMR systems where voting, temporal and state synchronization, as well as the exclusion of erroneous and the integration of repaired units can be realized automatically. Besides general problems like voting, there are in a typical application also other problems that can not be solved in a generic way. One typical example is the emergency stop, where a safe shutdown of the system must be guaranteed. To support the developer however, it is possible to model these application dependent mechanisms within our tool. Tests can be modelled in multiple ways to detect faults. The reaction to detected faults can then be realized by the run-time system (e.g. excluding the erroneous unit in a TMR system or switching to the backup unit in a hot-standby system) or by switching the application mode to a user defined mode realizing the emergency stop.

Another example is the interaction with the environment. Typically, it is not possible to detect actuator errors immediately. Rather, the validity must be tested by observing the reactions of the controlled system. This can be realized by the specification of appropriate tests within our tool.

Development Status and Future Work: We are currently implementing the code generator in collaborating with the German certification authority TÜV. A previous version of our code generator limited to the design of triple modular redundancy systems[1] showed the potentials of our approach: up to 95% of the whole code could be generated automatically with our tool. The approach will be tested in several lab applications, e.g. an elevator control. In addition, the application of our tool in an industrial project is planned for the end of 2007.

[2] By the term platform, we understand the combination of hardware, operating system and programming language

References

[1] C. Buckl, A. Knoll, and G. Schrott. The Zerberus Language: Describing the Functional Model of Dependable Real-Time Systems. In *Dependable Computing, Second Latin-American Symposium, LADC 2005, Salvador, Brazil, October 25-28, 2005, Proceedings*, Lecture Notes in Computer Science, pages 101–120. Springer, Oct. 2005.

[2] C. Buckl, M. Regensburger, A. Knoll, and G. Schrott. Models for automatic generation of safety-critical real-time systems. In *Second International Conference on Availability, Reliability and Security (ARES 2007)*, pages 580–587. IEEE Computer Society, Apr 2007.

[3] H. Kopetz and G. Bauer. The Time-Triggered Architecture. *Proceedings of the IEEE*, 91(1):112 – 126, Jan. 2003.