Synthesis of Diagnostic Techniques Based on an IEC 61508-aware Metamodel

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

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

***Abstract*—Safety standards, such as IEC 61508, play an important role in assuring the safety of embedded systems. Since model-driven development (MDD) is also gaining importance in the development process of these systems, an integration of the standards with existing modeling theory is promising. However, one of the basic building blocks of MDD, the metamodels, have not been made "standard-aware" yet. This paper presents a first step of such an integration by using a standard-aware metamodel to synthesize diagnostic techniques. This is an important task, because the correct selection and implementation of these techniques is traditionally a manual, labor-intensive task. The necessary steps of such an integration are discussed, including the definition of the metamodel, the formulation of an algorithm to select the right diagnostic techniques, and the implementation of code generation.**

I. Introduction

During the last years, model-driven development (MDD [7]) has become a very popular concept in some areas of software engineering and it is also experiencing an upswing in the development of safety-critical systems. These systems are traditionally designed and implemented with great care and according to application specific safety standards, whose compliance is checked by certification authorities like the Federal Aviation Administration or the European Aviation Safety Agency to assure safety. However, commonly used safety standards like IEC 61508 [4] or RTCA DO-178B [8] were published before the rise of MDD and model-driven approaches are not mentioned there. On the other hand, safety-critical systems were also not the main focus in the development of MDD, so no generally accepted way exists as to how MDD should be handled in the development and certification process of safety-critical embedded systems. One of the key challenges is that the main building blocks of MDD, the metamodels, are not aware of the concepts of safety standards, so within this paper, we present an approach how the fault models of IEC 61508 can be integrated into a metamodel to automatically synthesize diagnostic techniques.

The first contribution of this approach is a standard aware metamodel, which presents an easy way to use the fault models from the IEC 61508 in a MDD process. This metamodel can be used even by developers without much knowledge of the standard. The second contribution of this approach is an automatic selection process for diagnostic techniques, based on the presented metamodel. Traditionally, this selection process has to be performed manually and is very time-consuming. The automation of this process helps on the one hand to synthesize a lot of diagnostic techniques automatically and shows the developer on the other hand, which diagnostic techniques still have to be implemented manually (e.g. because they are very application specific).

The structure of the fault models of the IEC 61508 will be explained in Sec. II. A running example will be introduced in Sec. III to visualize the details of the approach, an appropriate metamodel will be constructed in Sec. IV and the automatic process of selection appropriate diagnostic techniques will be explained in Sec. V. Concluding the paper, Sec. VI will present related work und Sec. VII will give an outlook on future work.

II. Background of this Work

The safety standard IEC 61508 "Functional safety of electrical/electronic/programmable electronic safety-related systems" handles all aspects of electrical, electronic and programmable electronic systems (E/E/PES), when being used for executing safety critical functions. It was published for the first time in 1998 and has been accepted by multiple standardization committees around the world in the following years. One of the authors' main goals was to create a very general safety standard that can be used as a foundation for application specific standards, e.g. ISO 26262 [5] in the automotive domain. Due to the wide scope of the standard, it deals in big parts with development process characteristics and therefore big parts of it are not relevant for constructive development methodologies like MDD. The constructive parts of the standard that were extracted into this work are fault models and test functions.

The IEC 61508 follows a common principle for safety standards and does not specify in a precise way, which actions have to be taken to make a specific application certifiable, so it does not constrain new ideas and innovations. The certifiability of a specific application has to be checked as the case arises. However, to decrease the development costs of safety critical systems, the standard's appendix lists a set of diagnostic techniques that can be used for fault detection.

978-1-61284-820-4/11/$26.00 ©2011 IEEE

	diagnostic coverage		
component type	low	medium	high
$component\ type_1$	fm_1	fm_2	fm_3
...	...	...	...
clock	sub- or super-harmonic	sub- or super-harmonic	sub- or super-harmonic
...	...	...	...
$component\ type_n$	fm_{n-2}	fm_{n-1}	fm_n

TABLE I
PRINCIPLE OF IEC 61508'S FAULT MODELS

Due to the IEC 61508 being already about a decade old, many modern system and software engineering techniques are not addressed in it, like MDD. However, modeling concepts in general are much older than MDD, especially fault models have been used for a long time in the development of safety critical systems [3]. The IEC 61508 specifies fault models, which have to be detected by a safe E/E/PES. In particular, it breaks E/E/PE systems down to a list of basic hardware component types, like CPUs, buses or different kinds of memory. For each of these hardware component types, it specifies a list of fault models that can occur in this component, like "stuck-at". Furthermore, it separates each of these lists into three sections, according to the frequency of occurrence [4] of the fault models, and specifies which of them have to be detected to achieve an overall diagnostic coverage of low (60%), medium (90%) or high (99%).

This principle is visualized in table I.$component\ type_1$ to $componenttype_n$ are the different hardware component types, specified by the standard and fm_1 to fm_n are different, but not necessarily disjoint sets of fault models. For each of the component types there exists a table with appropriate diagnostic techniques, which look like the example in table II.

III. RUNNING EXAMPLE

The running example in this paper will be an elevator control, depicted in Fig. 1. This system consists of two main controllers, which are connected to twelve field controllers via CAN. The main controllers are executed in hot-standby and perform voting on the results of their computations to detect internal failures. For all controllers (main and field) in the system, we assume that the following faults may occur:

- CPU faults (A)
- ROM faults (B)

As the field controllers handle the interaction with the physical system, we assume that the following faults may occur there additionally:

- IO faults (C)
- internal bus faults (D)

Finally, we assume that the CAN bus may fail silently (E). This example is intended to present the reader this paper's approach to synthesize diagnostic techniques. A decently complex application was chosen to show how this automatic process can be used in a real development process and how it can simplify the development work. The letters in brackets after every fault will be used later as an index to make referencing to a specific fault easier.

To keep the example comprehensible for the reader, we do not model the system in more detail than described here and seen in Fig. 1. In fact, the approach that is proposed in this paper does not need more concrete system models.

mechanism	maximum diagnostic coverage	reference to more detailed information
$mechanism_1$	dc_1	$text_1$
...	...	...
RAM-test "Galpat"	high	table A.5.3
...	...	...
$mechanism_n$	dc_n	$text_n$

TABLE II
EXEMPLIFIED LIST OF DIAGNOSTIC TECHNIQUES FOR ONE COMPONENT TYPE

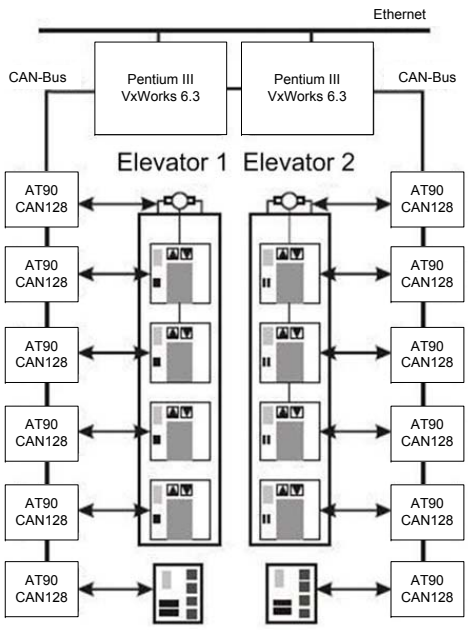

Fig. 1. Running example

IV. METAMODELING AND MODELING

This section will describe the general metamodel first and then go into detail how this metamodel can be used to model the application from the running example.

A. General Apprach

The metamodel for the integration of standard-compliant fault models has to be able to map the basic principle of IEC 61508's fault models, which has been presented in Sec. II. Therefore our metamodel is based on an abstract common superclass $FaultyComponentBehavior$, which describes the faulty behavior of a specific component in the system. From this superclass, we derive one specific non-abstract subclass for every component type that is listed in the IEC 61508, e.g. $FaultyCPUBehavior$ and $FaultyRAMBehavior$. Each of these subclasses gets an attribute for every subaspect

(e.g. $FailureRegisters$ and $FailureAddressCalculation$ for $FaultyCPUBehavior$) that is listed in the standard. Each of these attributes is of a unique enumeration type, which list the respective fault models. An excerpt of the whole metamodel is visualized in Fig. 2.

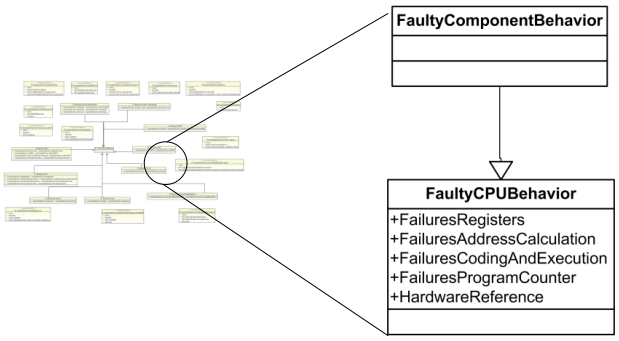

Fig. 2. Excerpt of the Metamodel

B. Application on the Running Example

To make the application, presented in Sec. III, safe, we have to handle every fault of the system's fault hypothesis. It is exceptionally important to notice, that typically many possible faults in safety critical systems are already handled by the system architecture, e.g. by the use of multiple channels. Only faults that are not yet covered by these architectural measures have to be detected by the use of explicit diagnostic techniques.

The general fault hypothesis of the example system is the following:

- main controllers: $\{A, B\}$
- field controllers: $\{A, B, C, D\}$
- CAN bus: $\{E\}$

However, due to the hot-standby structure of the two main controllers, only the following, reduced fault hypothesis has to be handled by explicit diagnostic techniques:

- main controllers: $\{\}$
- field controllers: $\{A, B, C, D\}$
- CAN bus: $\{E\}$

So we annotate the CAN bus with a $FaultyNetworkBehavior$, whose attribute is set to $NoTransmission$ to deal with E. Moreover, we annotate every field controller with various subclasses of $FaultyComponentBehavior$:

- $FaultyCPUBehavior$, attribute $codingAndExecution$ set to $wrongCoding$ (refering to A)
- $FaultyROMBehavior$, attribute set to $StuckAt$ (refering to B)
- $FaultyDCBehavior$, attribute $digitalIO$ set to $StuckAt$ (refering to C)
- $FaultyBusBehavior$, attribute $general$ set to $StuckAt$ (refering to D)

This model will be the input for the automatic selection of diagnostic techniques.

V. Test Selection

This section will present the process of selection appropriate diagnostic techniques. First, it will show the general approach and then it will go into detail how this can be used for the running example.

A. General Approach

This selection process exploits, again, the structure of the IEC 61508 fault models. Especially, it exploits the relationship between diagnostic techniques and faults, which was depicted in Fig. I and Fig. II.

The standard does not link diagnostic techniques and faults directly with each other, but uses an intermediate step, which is based on the diagnostic coverage (DC) of diagnostic techniques. The first step to select an appropriate diagnostic technique for a modeled fault is therefore to transform the model into a more abstract intermediate model, which maps the enumeration attributes of each failure to one of the integers 1, 2 or 3, representing the required diagnostic coverage (low, medium or high) for a specific system component.

Then, a request can be put to a library of diagnostic techniques, if there is a diagnostic technique available, which can deliver the desired diagnostic coverage for the specific system component.

If an appropriate diagnostic technique is available, the code generation capabilities of MDD can be used to synthesize it. An in-depth analysis of the IEC 61508 has shown that a lot of diagnostic techniques can be implemented in a general way and stored in a library. Should no appropriate diagnostic technique be available after all, the user can still be informed in a textual way, which algorithm has to be implemented manually, because the IEC 61508 offers a complete list of algorithms.

It has to be noted that obviously the scheduling of the synthesized diagnostic techniques is an important problem, regarding the fitting in the system's schedule as well as the diagnostic technique's execution frequency. In this paper, we assume that there is an oracle available which can solve this problem, however we have already conducted some research on it which will be published in the future.

B. Application on the Running Example

Applying the selection workflow to the model, defined in IV-B, we derive that we need the following diagnostic techniques on every field controller:

- CPU test, DC 1
- ROM test, DC 1
- IO test, DC 1
- Internal bus test, DC 1

Moreover, we need a network test for the CAN bus with DC 1.

A request to our library of diagnostic techniques delivers the following result for the field controllers:

- CPU test, DC 1 $\Rightarrow$ self-test by software, limited number of patterns (source code template available)

- ROM test, DC 1 $\Rightarrow$ signature of one word (source code template available)
- IO test, DC 1 $\Rightarrow$ failure detection by online monitoring (no source code template available)
- Internal bus test, DC 1 $\Rightarrow$ failure detection by online monitoring (no source code template available)

This result means that three diagnostic techniques are required to handle the four faults, as IO faults and internal bus faults can be handled by the same mechanisms. Moreover, the source code of two of these mechanisms can be generated automatically. Only the failure detection by online monitoring has to be implemented manually.

A request to our library of diagnostic techniques delivers the following result for the CAN bus:

- network test, DC 1 $\Rightarrow$ test pattern (source code template available)

This result means, that an additional diagnostic technique is required to handle the faults of the CAN bus. These mechanisms can also be synthesized automatically.

VI. Related Work

MDD of safety critical embedded systems is not a completely new research topic, but only a few other research groups try to integrate safety standard related concepts into their tool chains for code generation. Moreover, to the best of our knowledge, no other research group explicitly separates fault detection and fault handling. A very common approach is the avoidance of the conventional sequence *fault detection* followed by *fault handling* by using *fault masking*, where multiple variants of every software function are executed in parallel and the general result is defined by a vote over all variants' results.

The MDD tool "SynDEx", which is described as a system level CAD software that is based on the "Algorithm Architecture Adequation" methodology [6], implements this fault masking approach. It possesses a mechanism for the automatic generation of safety critical software, which is based on the following work flow: a model of the system's hardware is used as a foundation on whose top tasks are automatically replicated and distributed. A detailed description of this clear implementation of fault masking can be found in [2].

The main advantage of our concept, compared to fault masking, is a much higher flexibility in the generated system's software and safety architecture. Even if task replication and fault masking is a standard compliant way of building safety critical systems, it is very expensive with respect to hardware costs. In many cases, fault tolerance can be achieved by cheaper means, particularly in systems with low safety requirements (e.g. SIL 1).

A lot of work on fault modeling is done in the area of safety analysis. In this field, various approaches use error modeling not for the actual source code generation, but for the analysis of existing systems. These analyses may focus on the search for bottlenecks in achieving dependability requirements, for example by using probabilistic timed automata in which the nodes represent error states and the edges denote transition probabilities. [1].

Another research area that is related to our work tries to improve the line of communication between the safety engineers and the software engineers [9]. In contrast to our work, no further artifacts are generated from the system models there, but these models are only used as a communication tool between the different kinds of engineers.

VII. Conclusion and Future Work

This paper presented our approach for the synthesis of diagnostic techniques for IEC 61508. It is based on a standard-aware metamodel for the description of hardware faults and an automatic process for the selection and generation of appropriate diagnostic techniques. The approach aims at making the development of standard-compliant safety-critical systems easier by shifting parts of the required knowledge from a developer into a development tool.

As already mentioned in Sec. V, the scheduling of generated diagnostic techniques is a non-trivial problem. We already started to investigate this problem and will publish the results in the future.

Another topic for future work is a tighter coupling of our approach with standard compliant development processes, with respect to documentation. Documentation is a key aspect to achieve standard compliance in a development process. The ultimate goal is usually to document that every step in the whole process has been planned, executed und reviewed. To make our approach even more suitable, we will determine which parts of this documentation can be generated automatically from our models.

References

[1] Hseyin Aysan, Sasikumar Punnekkat, and Radu Dobrin. Error modeling in dependable component-based systems. *32nd Annual IEEE International Computer Software and Applications Conference*, pages pp. 1309–1314, 2008.

[2] Alain Girault, Hamoudi Kalla, Mihaela Sighireanu, and Yves Sorel. An algorithm for automatically obtaining distributed and fault-tolerant static schedules. *Proceedings of the 33th International Conference on Dependable Systems and Networks*, page pp. 159, 2003.

[3] H. Hölscher and J. Rader. *Microcomputers in safety technique. An aid to orientation for developer and manufacturer.* Verlag TÜV Rheinland, 1984.

[4] International Electrotechnical Commission. 61508, functional safety of electrical/electronic/programmable electronic safety-related systems, 1998.

[5] International Organization for Standardization. DIS 26262, road vehicles functional safety, 2009.

[6] C. Lavarenne, O. Seghrouchni, Y. Sorel, and M. Sorine. The SynDEx software environment for real-time distributed systems, design and implementation. *Proceedings of the European Control Conference*, pages pp. 1684–1689, 1991.

[7] OMG. Model driven architecture, a technical perspective. Technical Report No. ab/2001-02-04, Object Management Group, 2001.

[8] Radio Technical Commission for Aeronautics. DO-178B, software considerations in airborne systems and equipment certification, 1992.

[9] Gregory Zoughbi, Lionel Briand, and Yvan Labiche. A uml profile for developing airworthiness-compliant (RTCA DO-178B) safety-critical software. Technical Report TR SCE-06-19, Software Quality Engineering Laboratory (SQUALL), 2006.