Developing safety-critical mechatronical systems

Matthias Güdemann, Frank Ortmeier, Wolfgang Reif

Angaben zur Veröffentlichung / Publication details:

Güdemann, Matthias, Frank Ortmeier, and Wolfgang Reif. 2008. "Developing safety-critical mechatronical systems." In *Self-optimizing Mechatronic Systems: Design the Future; 7th International Heinz Nixdorf Symposium, 20-21 February 2008*, edited by Jürgen Gausemeier, Franz-Josef Rammig, and Wilhelm Schäfer. Paderborn: Heinz Nixdorf Institut.

Developing Safety-Critical Mechatronical Systems

Matthias Güdemann, Frank Ortmeier and Wolfgang Reif
Chair of Software Engineering and Programming Languages
University Augsburg
Universitätsstr. 14
86135 Augsburg
0821 598 2176
{guedemann,ortmeier,reif}@informatik.uni-augsburg.de

Summary

Developing high-assurance systems is always a challenging task. This is in particular true for safety-critical mechatronical systems. For these systems it is not only necessary to develop efficient software, which must often run on processors with limited resources but also to take carefully into account what environment is to be controlled and how this environment can be monitored.

Esterel Technologies SCADE Suite is a state-of-the-art development tool for safety-critical software. It is widely used in avionics and space applications. In this paper we show, how a model driven approach for software development can be used for mechatronical systems and what benefits can be achieved compared to traditional development processes. We illustrate the process on a real world case study: the height control system of the Elbe-Tunnel in Hamburg.

Keywords: safety-analysis, embedded systems, model driven

1 Introduction

The way software is being developed has dramatically changed in the last years. Traditional coding and programming has been significantly reduced and is now in many domains supported by automatic code generation. This allows for increased effort during design and analysis stages. The result of these new processes is, that software products are faster to develop and better reflect the business processes of the users they are built for. In the domain of mechatronical and embedded systems these new process are only very rarely used. Most of the software in such applications is still programmed in a traditional manner. This results in extended need for testing and a lot of – a posteriori – change requests.

In this paper we will show how model driven development processes can be used in the domain of mechatronical systems. We will use an interesting real world example for illustration and show what benefits this approach has compared to a traditional one. In Sect. 2 we will briefly describe Esterel Technologies' SCADE Suite and how system development is being done with this tool. Sect. 3 informally describes the case study and Sect. 4 illustrates a model driven development process on this example. Conclusions are summarized in Sect. 5.

2 Developing Safety-Critical Systems

The basic idea of model-based development of software is to top-down build executable models and refine them stepwise. The big advantage is, that this allows for very early evaluation and thus easy to correct models. In SCADE, models are described as a combination of state charts and data flows. In Sect. 2.1 we give a brief introduction to SCADE's syntax and semantics, while Sect. 2.2 describes how executable models for embedded and mechatronical systems may be built.

2.1 Semantics and Syntax of SCADE Models

The semantics of SCADE models is based on data flows[1]. Each single data flow can be seen as a sequence of values for a variable. The set of all possible data flows defines the semantics of the model. So semantically, this modeling language is very similar to the set of traces defined by a Kripke structure (which is for example used for many finite automata based languages).

[1] Technically, SCADE models are semantically grounded on LUSTRE. Detailed informations may be found in [4,5,6].

On the other hand specification in SCADE is very different. Every SCADE model has a fixed set of input and output variables. Input variables are marked with an arrow with a vertical line (see a, b, and c in Fig. 1) and output with a normal arrow (see o in Fig. 1). SCADE models are built of blocks. Each block has a fixed input and output interface. Basic blocks are composed to larger models by connecting outputs to inputs. A syntactic convention is that all inputs are drawn on the left side of a block and all outputs are drawn on the right side. System inputs may be connected to any block's input and system outputs can be any block's output. Note, that input data is processed *immediately* throughout the whole model. This is different to many other modeling languages, where information is propagated step-by-step through different components. To avoid inconsistencies direct feedback is not allowed[2]. However, there exists a special operator **FBY**, whose output is the input of the last step[3]. This operator is often used if data feedback is needed. A simple SCADE model is shown in Fig. 1.

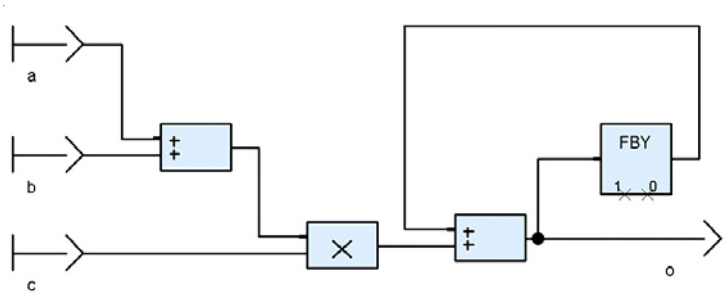

Figure 1: A simple SCADE model

This model has three inputs **a**, **b** and **c**. The first two inputs are added (``+"-operator) and multiplied (``*"-operator) with the third input. The output **o** is the accumulation of all previous results (``FBY"-operator). An example data flow $\mathbf{(a,b,c,o)}_i$ of the system is:

$$(1,2,3,9);(1,2,2,15);(2,2,2,23);...^{4}$$

SCADE also allows for embedding state machines in single blocks. Here, the semantics is that the state machine executes *exactly* one step for every step of the

[2] This is checked by a syntactic analysis.

[3] For the initial state the output of this operator must be defined explicitly (for more details see the documentation of SCADE).

[4] Because: (1+2)*3 + 0=9; (1+2)*2+9=15; (2+2)*2+15=23

data flow. Note, that SCADE models must always be deterministic. So modeling of failures – which typically occur non-deterministically – requires a little trick.

2.2 Building Executable Models of Safety Critical Systems

For building an executable system three things must be modeled: (1) the system itself, (2) the important part of the environment and (3) the possible failure modes. Note that (2) and (3) can also be put together in one category, but experience has shown that it is very convenient to split modeling of an ideal environment and failures into two phases. Fig. 2 sketches this structure[5].

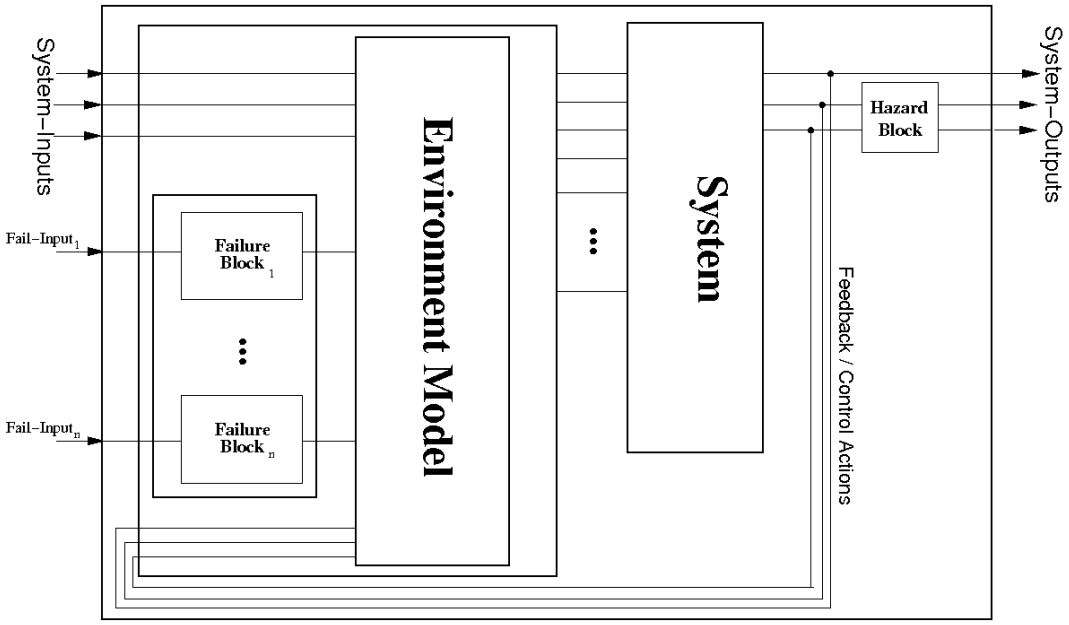

Figure 2: Basic structure for an executable SCADE model

Building a SCADE model of the system itself is very much similar to designing and implementing the system in a high-level, graphical design language. When using SCADE the whole system can then be encapsulated into one block, which reacts on environment inputs and generates control outputs. The next step is to build a model of the environment. This can also be encapsulated in a SCADE block, which will ultimately produce as outputs the necessary inputs for the functional system and will in some way react on the control measures of the system.

[5] Note, that FBY operators have been left out for better readability.

The inputs of this component are necessary for indeterminism. As stated above, all SCADE models must be deterministic. The only possibility for indeterminism is to introduce unspecified input variables. Note, that it is also possible to build a "chaotic" environment. This means the inputs for the control system are not restricted at all. However this will not yield good analysis results for the system under development in most practical applications.

When modeling failures, two different aspects are important. On the one hand it must be modeled when and how failures occur (e.g. transient or persistent), on the other hand it must be modeled what direct effect this failure will have (e.g. missed detection of a sensor or loss of a communications package). The first aspect is called occurrence pattern the later one is called the direct effects of the failure mode.

Occurrence patterns of failure modes are modeled in specific failure mode blocks. The outputs of these blocks are used in the SCADE model to trigger the direct effects in the system. Additionally the safety critical event (or hazard), which must be prohibited, is modeled as a SCADE block which aggregates system outputs. This allows for automatic check whether the hazard may occur or not. For the automatic check, SCADE has a rudimentary model checking [2] component called *Design Verifier* implemented. This modeling approach is very intuitive and can be easily understood, when taking a look at the example in Sect. 3.

3 An Example

As an example for developing and analyzing a safety-critical mechatronical system, we will present the height control system of the Elbe-tunnel in Hamburg.

The Elbe-tunnel is a road tunnel, which connects the harbor with the city of Hamburg. The old tunnel consisted of three tubes with two lanes each. This tunnel has been enhanced in late 2002 with a new fourth tube. The tunnel comprises a very complex control system which contains traffic engineering aspects like dynamic route control, locking of tunnel tubes, etc. We will consider only a small part of the whole system – the height control – in this paper.

3.1 The Elbe-Tunnel's Height Control System

The new tube has been built larger than the old tubes. This allows overhigh vehicles carrying goods from the harbor to use the tunnel to reach the city. An overhigh vehicle is a vehicle whose overall height is greater than 4.5 meters. The height control system must assure, that no collisions of overhigh vehicles with the tunnel's ceiling occur i.e. it must assure, that such vehicles only enter the correct tube.

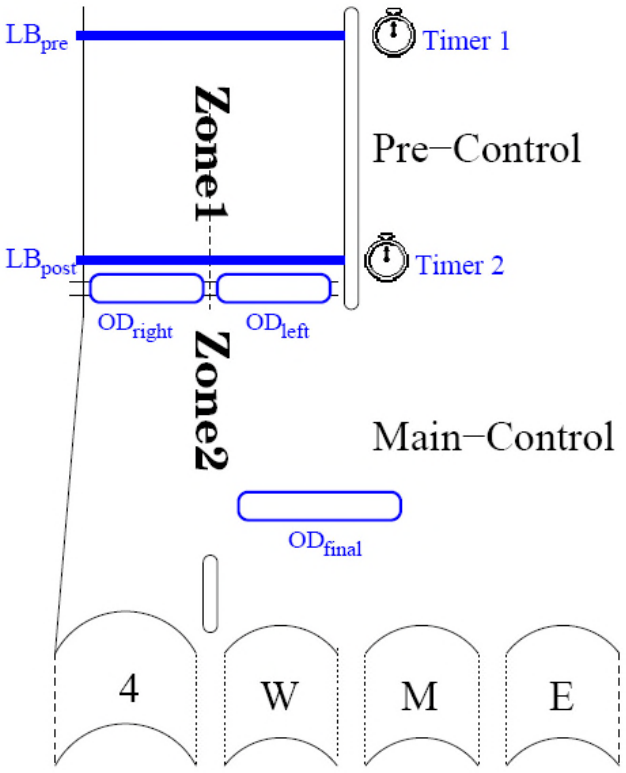

Figure 3: Layout of the northern entrance of the Elbe-Tunnel

In the following, it is necessary to distinguish between *high vehicles* (HVs), which may drive through all tubes and *overhigh vehicles (O*HVs), which may only drive through the new, fourth tube. Figure 3 sketches the layout of the tunnel and the planned sensors for the height control system. The fourth tube may be cruised from north to south and the east-tube from south to north only. We focus the analysis on the northern entrance, because OHVs may only drive from north to south.

The system uses two different types of sensors. Light barriers (LB) are scanning all lanes of one direction to detect, if an OHV passes. For technical reasons they cannot be installed in such a way, that they supervise only one lane. Therefore overhead detectors (OD) are necessary to detect, on which lane HVs and OHVs pass. ODs can distinguish vehicles (e.g. cars) from high vehicles (e.g. buses, trucks), but not HVs from OHVs (but light barriers can!). If the height control detects an OHV heading towards a different than the fourth tube, then an emergency stop is signaled, locking the tunnel entrance.

The idea of the height control is, that the detection starts, if an OHV drives through the light barrier LB_{pre}. To prevent unnecessary alarms through faulty triggering of LB_{pre}, the detection will be switched off after expiration of a timer (30 minutes). Road traffic regulations require that after LB_{pre} both HVs and OHVs have to drive on the right lane through tunnel 4. If nevertheless an OHV drives on the left lane towards the west-tube, detected trough the combination of

LB_{post} and OD_{left}, an emergency stop is triggered. If the OHV drives on the right lane through LB_{post}, it is still possible for the driver to switch to the left lanes and drive to the west- or mid-tube. To detect this situation, the height control uses the OD_{final} detector. To minimize undesired alarms (remember, that normal HVs may also trigger the ODs), a second timer will switch off detection at OD_{final} after 30 minutes. For safe operation it is necessary, that after the location of OD_{final} it is impossible to switch lanes. Infrequently, more than one OHV drives on the route. Therefore the height control keeps track of several OHVs. A formal specification of this system using finite automata may be found in [6,7].

3.2 Primary Failure and Hazards

There are two different, interesting hazards for the Elbe-Tunnel's height control: the collision H_{Col} of an OHV with the tunnel entrance and the tripping of a false alarm H_{FA}. For this paper only the hazard collision is taken into account. The model can be easily extended to also contain false alarms. There exists a variety of failure modes, which can be taken into account. Besides the obvious detector errors, another error comes into play. An OHV may need more than the upper bound of 30 minutes to travel through one of the zones. This may be caused by a traffic jam. This is not a failure in the traditional sense, but rather an unexpected behavior of the environment. However, from a logical point of view it may be treated in the same way as a (hardware) component failure. Finally, the misbehavior of high vehicles must be taken into account.

The component failures, which are considered in this case study – may be divided into four different groups:

1. False detection (FD)
   The sensor *does* indicate a vehicle, although there is *none*. Possible for all sensors.
2. Miss detection (MD)
   The sensor *does not* indicate a vehicle, although there is *one*. Only possible for OD-type sensors.
3. Overtime (OT)
   Actual driving time of an OHV exceeds the runtime of a timer. Possible for zone 1 and zone 2.
4. High vehicles (HV)
   A high vehicle beneath an overhead detector is interpreted as an OHV.

We write FD_{final} as abbreviation for false detection at overhead detector OD_{final} and analogously for all other sensors. Overtime failures can occur in zone 1 and zone 2. We write OT_1 resp. OT_2. Note, that the last item (HV) is not a failure in

the traditional sense, as overhead detectors can not distinguish between high vehicles and OHVs, high vehicles at the location of the sensors are (incorrectly) interpreted as OHVs. For the control system this has the same effect as a FD of the sensor. Traffic regulations require high vehicle to drive on the right lane. Because of this we introduce HV-type error only for OD_{left} and OD_{final}. High vehicles at OD_{right} are of course modeled (and may trigger the detector), but are not a failure as they are part of the expected working environment of the system. So altogether 12 failure modes are taken into account: $\{FD_{right}, MD_{right}, MD_{left}, MD_{left}, HV_{left}, FD_{final}, MD_{final}, HV_{final}, OT_1, OT_2, FD_{pre}, FD_{post}\}$.

4 A SCADE Model

How can this system be modeled in SCADE? The top level system model looks exactly like in Fig. 2. The environment block contains models of OHVs and HVs. These are very easy to build and basically model the physical properties of the vehicles i.e. (1) they move continuously, (2) they move in one direction and (3) they may be travelling at different speeds. In addition 12 failure mode blocks (see above) have been integrated in the model. Fig. 4 shows an exemplary failure mode block (for false detection at the first light barrier: FD_{LBpre}) on the left and a model of it direct effects on the right.

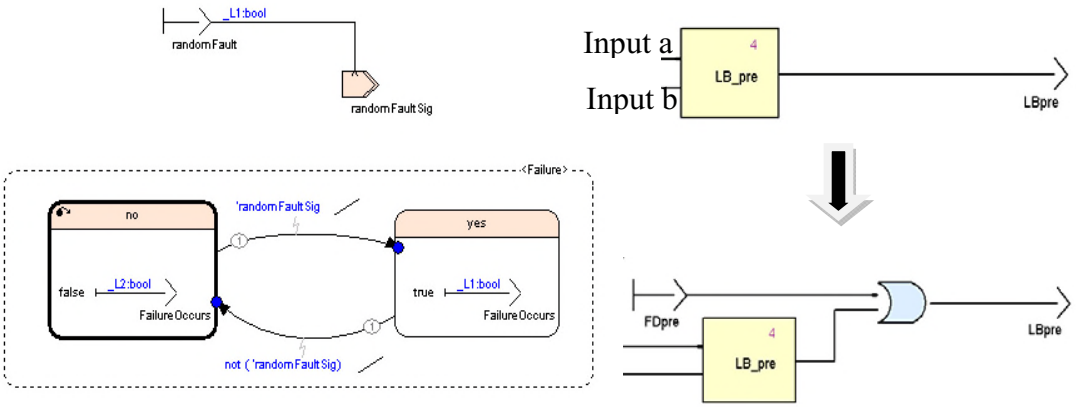

Figure 4: A failure mode block for the occurrence pattern of a false detection at LB_{pre} and its direct effect

Failure mode blocks describe how and when the failure may occur. They typically contain a state machine with two sates (“yes” and “no”). In the example, the failure mode may occur non-deterministically. Therefore transitions may be taken randomly at any time. This indeterminism is modeled with the input signal “randomFault”. The output of this block (“FailureOccurs” in Fig. 4 left) is then fed into the environment model to exhibit the effect of the failure. One such effect is shown on the right side of Fig. 4. In the top part of the figure, the initial part of the environment model of the first light barrier is shown: LB_{pre} will be emitting a sig-

nal, if the ray of light is interrupted (Input a) and if it is activated (Input b). In the lower part of the figure, the direct effects of the failure mode (FD_{pre}) are modeled also. Here, the light barrier component signals something if either the conditions described above hold *or* if a false detection occurs. The output of the light barrier LB_{pre}'s is either true because an overhigh vehicle is at the position of the light barrier (and interrupts it) or because of a false detection of this component i.e. FD_{LBpre}.

The model of the control system is shown in Fig. 5. It basically consists of two timers (TI1 and TI2), that activate the OD detectors (digital::count_down) and a counter (OHVcounter), that keeps track of how many OHVs are in the zone between LB_{pre} and LB_{post}. If the timer TI1 is timed out, then the counter is reset to 1 if at the same time there is a detection at LB_{pre} or to 0 otherwise.

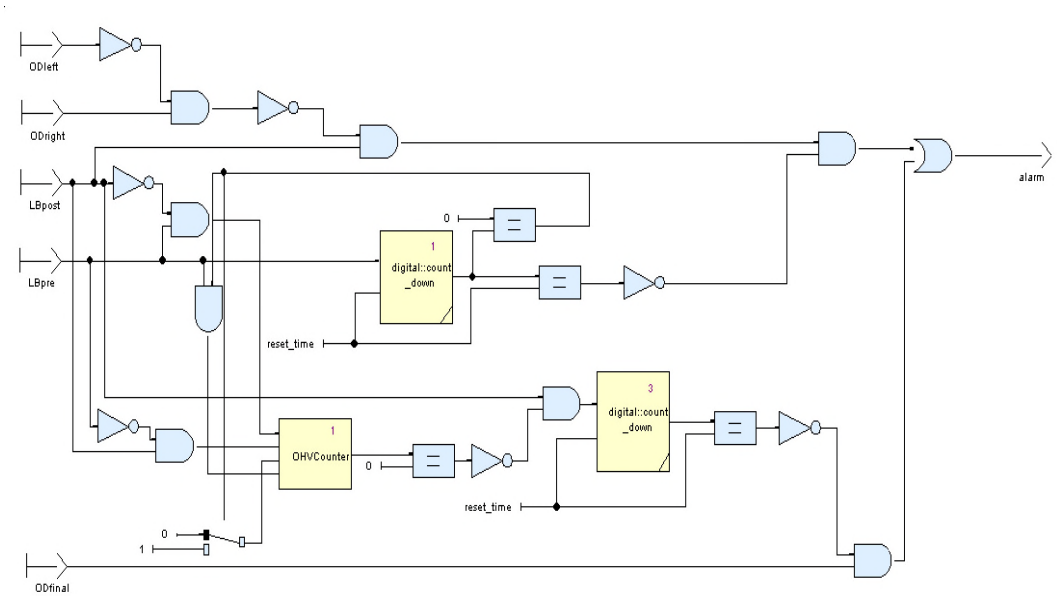

Figure 5: The control system

The system works as follows: It continuously monitors the first light barrier LB_{pre}. If a signal is detected, then a timer (TI1) is (re-)started. While this timer is active, the system activates the second part of the control (LB_{post}, OD_{right} and OD_{left}). If the second light barrier (LB_{post}) detects a signal and OD_{right} detects the high vehicle on the right lane, then a second timer will be (re-)started[6]. While this second

[6] All other combinations of detections of the overhead detectors, while LB_{post} detects an OHV will immediately trigger an emergency stop. This conservative approach has been chosen, because

timer (TI2) is active, any detection of the sensors OD_{final} will trigger an emergency stop. The second part of the control system will be active as long as either (1) TI1 expires (this means there has probably been a false detection at LB_{pre}) or until no more vehicles are between LB_{pre} and LB_{post}.

Results:

This model – system block together with environment and failure mode blocks – may now be used for analysis and testing. Model checking can be directly done using SCADE's built in model checker – the *design verifier*. And it is also possible to evaluate the system design with the simulation engine.

For simulation runs, the user triggers input variables at his choices (this means in the example starting and restarting OHVs and HVs). The simulation then shows, that the control system in principle works as it should, the runtimes of the timers are correct wrt. the environment specification and that the planned combination of sensors works.

Bounded model checking can then be applied to rigorously check all scenarios up to a certain maximum length. In the example, it turned out that the planned systems had a fundamental design error. The problem arises if two OHVs pass the first light barrier *simultaneously*. The system will then only recognize one. This alone is not safety-critical. If however, both vehicles drive at very *different speeds* (one very fast and the other very slow). Then the second part of the control will be deactivated (by the first OHV, as the OHV counter is decremented to zero). If the second OHV continues driving at low (but still acceptable) speed, then it is possible, that TI2 has timed out *before* the second OHV reaches OD_{final}. As a consequence, OD_{final} will be deactivated when the second OHV passes the detector and a collision might occur. Note, that this scenario *does not* involve any component failures – it represents a major design flaw. This very improbable scenario was not foreseen by the engineers. It can also only be found by very extensive testing. A possibility to fix the problem is, to either never stop TI1 before it times out or to add additional overhead detectors at LB_{pre}.

Unfortunately, in SCADE – due to the complexity of the model – only bounded model checking possible for this case study. So it can only be used to find/identify design flaws and not for showing the absence of design flaws. However, it has been shown, that the presented flaw is the only design error. In [7], this example is translated into a specific model checking language for the SMV tool. This symbolic model checker is capable of complete state space exploration for this model.

---

wrong driving OHVs should be stopped as soon as possible. Therefore some additional false alarms are accepted in exchange for early emergency stops.

5 Conclusion

Developing safety-critical system was and will always be a difficult task. Using a model-based approach can help and support early detection of design flaws and errors. A model-based approach is in particular useful, if code generation is possible. Esterel Technologies' SCADE Suite is a modern software development tool for highly safety critical systems. It not only comes with a built in verification and simulation engine but also contains a certified code generator for various target platforms.

In this paper we showed, how SCADE Suite can be used for model-based development of a mechatronical system. The approach allows for separate modeling of functional aspect of the system, the environment and errors. If these three parts are put together, then either SCADE's design verifier or the simulation engine may be used to validate and verify the design of the system at early stages. In the presented example this approach was applied to a traffic control scenario. A design flaw was identified and some evaluations of failure tolerance were done.

In summary, we think that this approach for system development is easy and flexible and results in high quality of the systems. This also corresponds with our experience and the observation, that testing and adaptations are often the major cost factor in the development of mechatronical systems.

Literature

[1] Parosh Aziz Abdulla, Johann Deneux, Gunnar Stalmarck, Herman Agren, and Ove Akerlund. Designing safe, reliable systems using SCADE. In Proceedings of ISOLA'04. Springer-Verlag, 2004.

[2] D. A. Peled E. M. Clarke Jr., O. Grumberg. Model Checking. The MIT Press, 1999.

[3] A. Benveniste, P. Caspi, S. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone, "The synchronous languages 12 years later," Proceedingsof the IEEE, vol. 91, no. 1, January 2003.

[4] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, "The synchronousdata-flow programming language LUSTRE", Proceedings of the IEEE Vol. 79, September 1991.

[5] P. Caspi, A. Curic, A. Maignan, C. Sofronis, S. Tripakis, and P. Niebert, "From simulink to scade/lustre to tta: a layered approach for distributedembedded applications", Proceedings of LCTES '03, ACM Press, 2003.

[6] F. Ortmeier, W. Reif, G. Schellhorn, A. Thums, B. Hering, and H. Trappschuh. Safety analysis of the height control system for the Elbtunnel. Reliability Engineering and System Safety, 81(3), 2003.

[7] F. Ortmeier, W. Reif, and G. Schellhorn. Deductive cause-consequence analysis (DCCA). In Proceedings of IFAC World Congress. Elsevier, 2006.

Author

Dr. Frank Ortmeier is working as a senior research affiliate at the Chair of software Engineering and Programming Languages at the University Augsburg. His current research topics comprise various aspects of Organic Computing Systems, model-based design and analysis of safety-critical embedded systems and the combination of modern software engineering techniques in the context of robotic and mechatronical systems. He got his Ph.D. in 2005 for his work on formal safety analysis methods. A key innovation in this research was the development of a novel formal safety analysis method: Deductive Cause-Consequence Analysis (DCCA). He studied mathematics, physics and computer science at the University Augsburg. In 2001 he got his degrees in mathematics, physics and computer science at the University Augsburg.

Dipl. Inf. Matthias Güdemann studied computer science at the University Augsburg. He got his diploma in 2005 and is since then a research affiliate at the Chair of Software Engineering and Programming Languages. His research interests include Organic Computing, model-based design and analysis of safety critical systems and the application of formal methods in engineering domains.

Prof. Dr. Wolfgang Reif is professor for Software Engineering and Programming Languages at the University Augsburg. He was appointed to the chair in 2000. Before that he was a professor at the University Ulm. His group is very well known in the domain of interactive theorem proving and maintains and develops the KIV interactive theorem prover. Current research topics include Organic Computing, Secure E-Commerce and safety-critical systems. He is also dean of the faculty of applied computer sciences at the University Augsburg.