VTT Technical Research Centre of Finland

Verification of fault tolerant safety I&C systems using model checking

Pakonen, Antti; Buzhinsky, Igor

Published in:
2019 IEEE International Conference on Industrial Technology (ICIT)

DOI:
10.1109/ICIT.2019.8755014

Published: 01/02/2019

Document Version
Early version, also known as pre-print

Verification of fault tolerant safety I&C systems using model checking

Antti Pakonen[1], Igor Buzhinsky[2, 3]
[1] VTT Technical Research Centre of Finland Ltd., Espoo, Finland
[2] Department of Electrical Engineering and Automation, Aalto University, Espoo, Finland
[3] Computer Technology Department, ITMO University, St. Petersburg, Russia
antti.pakonen@vtt.fi, igor.buzhinskii@aalto.fi

***Abstract*—Model checking has been successfully used for detailed formal verification of instrumentation and control (I&C) systems, as long as the focus has been on the application logic, alone. In safety-critical applications, fault tolerance is also an important aspect, but introducing I&C hardware failure modes to the formal models comes at a significant computational cost. Previous attempts have led to state space explosion, and prohibitively long processing times. In this paper, we present a method for adding hardware component failures and communication delays into I&C application logic models for the NuSMV symbolic model checker. Based on a case study built around a semi-fictitious, four-redundant nuclear power plant protection system, we demonstrate how even detailed system designs can be verified, if the focus is kept on single failure tolerance.**

***Index Terms*—formal verification, model checking, fault tolerance**

I. Introduction

Model checking [1] is a proven method for exhaustive verification of instrumentation and control (I&C) system's application logic—be it the software in a programmable logic controller (PLC) or the configuration of a field-programmable gate array (FPGA) circuit [2]. Yet, the focus is often on the logic specification, e.g., a function block diagram, alone. However, in reality, the application logic operates on hardware components, which are also subject to failure.

Single failure tolerance is often a requirement for safety-critical I&C systems—the failure of any individual component shall not prevent the system from performing its function. Tolerance to single failure can be achieved with redundancy. Redundant subsystems, together capable of performing the desired tasks even if one of them fails, are used in application domains such as nuclear energy [3], aviation [4], [5], aerospace [6], [7], railway [8], and automotive [9] industries.

Verifying that the failure tolerance mechanisms actually work calls for the modeling of both the application logic and the different failure modes of the underlying hardware. Furthermore, distributing the logic between redundant, separated systems introduces communication delays and asynchrony, which should also be addressed in the analyses. Previous attempts at using model checking to evaluate these aspects in one model have run into scalability issues. It has not been possible to analyze I&C system design of nearly the kind of complexity that is otherwise (with a logic-only model) not an issue for model checkers.

In this paper, we present a method for verifying the fault tolerance of I&C systems based on model checking. In addition to modeling the application logic, we also account for the failure modes of the underlying I&C system hardware components, and the communication delay between distributed computers. As the case example, we use the reactor protection system of the proposed U.S. EPR nuclear power plant. We are able to simplify the failure model by focusing strictly on the verification of single failure tolerance, as required by the Finnish regulatory guides on nuclear safety. We demonstrate our approach using the symbolic model checker NuSMV.

II. Failure tolerant I&C systems

A. Terminology

Below, we define some concepts related to failure tolerance, using the Finnish regulatory guides on nuclear safety and security (YVL) as a guideline.

Single failure criterion means that the system shall be able to perform its function even if any single component designed for the function fails. Protection against single failure is commonly achieved using several (potentially identical), redundant subsystems placed in physically separated **divisions**.

Consequential failure refers to "a failure caused by a failure of another system, component or structure or by an internal or external event at the facility" [10]. For example, a failure of a power supply system or a ventilation system can result in the subsequent total failure of several I&C system devices, and still be considered a single failure that shall be tolerated.

Common cause failure (CCF) refers to a "failure of two or more structures, systems and components due to the same single event or cause" [10]. Protection against CCF can be achieved using diverse backup systems (e.g., a different supplier, technology, or operating principle).

Passive failure means that the system fails to produce the required response. **Active failure** (or "spurious actuation" [11]) refers to inadvertent actuation without a real demand.

B. Example system: U.S. EPR Protection System

Our case example of a fault-tolerant I&C system is the reactor protection system of the proposed U.S. version of the European Pressurized Water Reactor (EPR) nuclear plant

[3][1]. Based on Areva NP's TELEPERM XS technology, the Protection System (PS) is organized into four redundant, independent *divisions*, located in separate buildings [12] (see Fig. 1).

The PS utilizes functional units called Acquisition and Processing Units (**APU**s) and Actuation Logic Units (**ALU**s) [12]. The APUs (1) acquire signals from the process sensors and monitoring systems via the Signal Conditioning and Distribution System (SCDS) using a hardwired connection, (2) perform calculations and setpoint comparisons, and (3) distribute the results to the ALUs for voting. The ALUs perform voting over processing results and issue actuating results, taking into account operator control actions from the Safety Information and Control System (SICS) user interface. The actuation orders are sent to the Priority and Actuator Control System (PACS) via a hardwired connection. The communication between the redundant divisions is relayed via a Profibus network, using fiber-optic cabling to achieve electric separation. [3]

Tolerance against a single failure (of an input sensor, a unit of the PS, or an actuator device) is based on the four-division structure, and selection and voting functions in the APUs and ALUs. Each signal in the PS logic, in addition to its value, has a **status**, which can be set to "fault" by failures detected by input modules or function processors. The status is then used to exclude invalid signals in selection (e.g., second-maximum[2], second-minimum) and voting (n-out-of-m) blocks. Even if a single failure disables an ALU logic performing the vote, the redundant ALUs can still actuate the reactor trip function. [3]

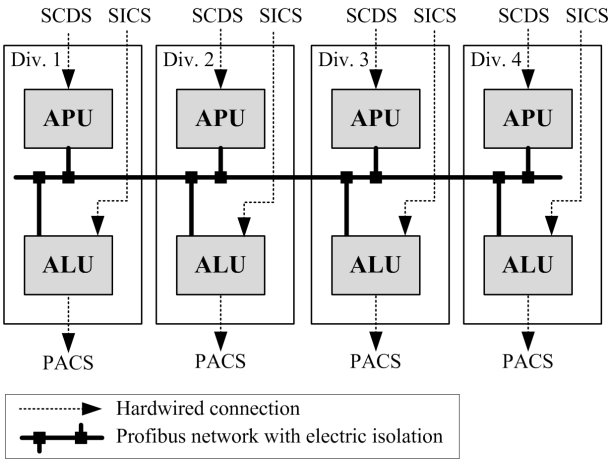

Fig. 1. Simplified architecture of the U.S. EPR Protection System

III. I&C LOGIC MODEL CHECKING

Model checking [1] is a formal, computer-assisted method that can be used to verify that a model of a (hardware or software) system fulfills given properties. The system model is typically expressed as a finite state machine. Formal properties describing wanted or unwanted behavior are specified using *temporal logic* languages like LTL, CTL [1] or PSL [13]. A model execution path violating a property is returned to the analyst as a *counterexample* trace, possibly revealing a design issue.

A key challenge is to avoid state space explosion, where the number of states to enumerate through becomes enormous [1]. *Symbolic model checkers* such as NuSMV [14] mitigate the problem using binary decision diagrams (BDD) [15] rather than explicit representation. By executing Boolean satisfiability (SAT) solvers, NuSMV is also capable of performing *bounded model checking* (BMC), where a limit is placed on the length of checked state transition sequences.

Model checking I&C application logics based on function block diagrams has been an active research topic for several years, with different areas of application [16]. It has been proven applicable for ensuring the correctness of industry-sized PLC programs [2], [17], but, nevertheless, is not yet a wide-spread industry practice in any application area.

I&C logics can be verified using either *open-loop* or *closed-loop* modeling. Open-loop model checking only considers the model of the "controller" logic, while in the close-loop model, feedback from the controlled plant is taken into account [18]. Closing the loop can help limit the state space of the model [18], but generating a realistic plant model can be a challenge. At the same time, limiting the controller model behavior might accidentally filter out scenarios relevant to safety, and analysis times can actually increase [19].

Since 2008, VTT has successfully used model checking to verify both early (functional) and detailed design of safety I&C systems for Finnish nuclear power plants [2]. A graphical tool called MODCHK [11], [20] is used to (1) manually define a collection of vendor-specific elementary function blocks, (2) model the function block diagrams with a graphical editor, (3) specify the properties with a text editor, (4) generate the necessary input files for NuSMV, and (5) visualize counterexamples produced by NuSMV with an animated view of the function block diagram.

In MODCHK, each signal has both a **value** (Boolean or integer) and a fault/validity **status** (a Boolean variable). The status processing is explicitly defined for each elementary function block, and the counterexample animation feature uses a dashed line to show a faulty/invalid signal [20]. In nuclear applications, the status processing feature has been very relevant, as it is not just used in TELEPERM XS (see Section II-B), but also in a similar way in Rolls-Royce's Spinline platform [21]. Status processing logic has also played a role in about 12% of the design issues VTT has identified [11].

Although successful, VTT's work has focused on the open-loop verification of the application logic (PLC software, or FPGA configuration), alone. The models do not account for (1) feedback from the controlled process, (2) failure modes of the underlying I&C system hardware, nor (3) communication delay and asynchrony in distributed applications. Omitting the

[1]No project is underway to construct an EPR in the U.S., but the U.S. Nuclear Regulator NRC has published parts of the plant supplier Areva NP's 2013 Final Safety Analysis Report (FSAR) for a suggested U.S. variant.

[2]a block selecting the remaining maximum number in a set, after excluding the maximum number

hardware failure modes, in particular, makes verifying the systems' failure tolerance impossible.

Previously explored methods to include hardware faults [22] and asynchrony [23] in the models have resulted in scalability issues that prevent their use in the verification of detailed I&C logic designs of realistic complexity.

IV. Related research

Several studies address failure tolerance in model checking. In [5], faults are added to a model of an aircraft wheel brake system with two redundant pistons, which is then verified with SCADE design verifier. In [6], fault variables are introduced in a state model of a 2-redundant spacecraft controller system, which is then verified with Spin [24]. In [4], NuSMV is used to verify whether an avionic altitude switch with three altimeters can tolerate measurement errors. In [7], the real-time model checker UPPAAL [25] is used to verify the fault tolerance of a 3-redundant aerospace system. In [8], the application is a 2-redundant railway interlocking system, and in [26], it is an FPGA logic. Finally, in [9] failure mechanisms are added to a NuSMV model of an automotive brake-by-wire system based on failure modes and effects analysis (FMEA) [27].

Other ways of combining FMEA and model checking have also been proposed. In [28] and [29], FMEA is supported by using model checking to identify the system-level consequences of component failures. In [30], probabilistic model checking is used to identify which components contribute most to system-level failures.

A common problem with many of the proposed methods is that the system model has to be kept very abstract, or the state space becomes too large [22], and/or the analysis cannot be performed in a reasonable time [26]. Instead of detailed system design, the models are based on "specified behavior" [6], "functional model" [9], or other abstraction or simplification.

When revealed, a typical model scale is 10^6 states [6], [8], [30]. In [28], the authors use a model with 10^9 states (but due to the necessary iterations, entire verification effort still takes days). Meanwhile, the complexity of the detailed design (no-fault) models VTT is verifying is on a wholly different level, often with 10^{20}, sometimes even 10^{30} reachable states.

V. Failure modeling approach

Instead of specifying a full failure model allowing all the processors and communication links fail (as in [22]), we can simplify the model (see Fig. 2) by keeping our focus on verifying *single fault tolerance* in *open-loop models*. To illustrate the idea, we use the U.S. EPR PS (see Section II-B) as an example:

1) First of all, it suffices to fully model only one division (APU+ALU). If the divisions are identical, any verification result for the included (non-failing, see below) ALU should hold for the redundant (non-failing) ALUs, as well.
2) There is no need to assume failures for the included ALU. The objective is to verify that each *non-failing* division will operate according to specification. In open-loop analysis, we are not interested in the outputs of the one ALU that can fail, and for the same reason the number of ALUs whose outputs are correct at each time instant is also not important.
3) It is sufficient, overall, to model the failures for one division, only. If the divisions are identical, we can assign the failures to any single division (other than the one that is fully modeled).

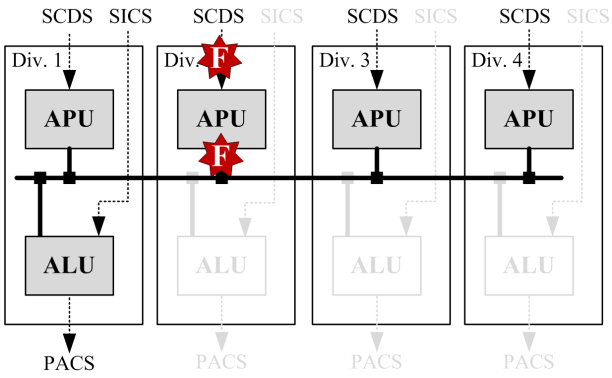

Fig. 2. By focusing on single failure scenarios in open-loop, the failure model can be made fairly simple.

Furthermore, instead of having a complex failure model for each component, it suffices to assume that:

1) Either a sensor device or the sensor—APU communication can fail on one division, in which case we replace the correct, measured value with a non-deterministic, arbitrary value at the sensors output.
2) Either the APU hardware or the APU—ALU communication link can fail on one division, in which case we replace the correct signal values with non-deterministic, arbitrary values at the APU outputs.

The failure model is implemented in NuSMV by inserting modules on one division to all signals (1) from the SCDS to the APUs and (2) from the APUs to the included ALU (see Fig. 2). Module `FAULT_BIN` is used for Boolean and `FAULT_ANA` for integer signals. At any time instant, the module can nondeterministically enter a fault mode, and replace the actual signal value with a nondeterministic variable. The status of the signal is also nondeterministic, meaning that the failure can be either *self announcing* (status = fault) or *non-self announcing*, i.e., not detectable by APU/ALU logic model. The actual fault status is separately output by both blocks, making it possible for the analyst to observe the failures even if they are non-self announcing.

Rather than analyzing in detail what the possible hardware failure modes are, we allow for all possible combinations of failures, including consequential failures. It might then be up to the analyst to determine if the failure combinations in a counterexample are actually possible. As we allow each signal in the failing division to fail independently and non-deterministically, some of the permitted scenarios can seem

unrealistically chaotic. E.g., the different outputs of a failing APU can permanently freeze in a state where some signals fail actively and others passively. Still, such a scenario is also feasible due to single failure, if, e.g., the CPU cooling fails.

VI. Delay modeling approach

We extend the approach described in Section V to account for delays and asynchrony. Since each APU and ALU operates on its own CPU, modeling delays within APU and ALU logic is unnecessary. A natural approach to asynchrony would involve modeling the ALU and APUs as different processes with interleaving executions, involving nondeterministic delays between the APUs and the ALU. Such an approach would be easy to implement in model checker Spin but not in a symbolic model checker such as NuSMV due to the synchronized execution of all modules of the formal model. Specifically for symbolic model checking, we instead assume that the executions of all APUs and the ALU are synchronous, but the signals entering and exiting APUs may experience nondeterministic delays which are bounded by the number of cycles $d_{\max}$.

The delays are implemented in the following way. Assume that $s_1, ..., s_k$ are signals whose delays must be synchronized, including fault/validity statuses of these signals. Since APU input signals are passed over different physical connections, they need not be synchronized, leading to $k = 2$ (a signal and its fault/validity status). In contrast, output signals of each particular APU are passed together. To account for both cases, an instance of a module `NONDET_DELAY` is added for each input signals of each APU and for outputs signals of APUs once per each APU. This module nondeterministically delays the signals passing through it with bound $d_{\max} = 1$ (i.e., by at most one cycle). This possible delay is applied synchronously to all k signals. The presence of the delay is controlled by a Boolean variable `delaying`, whose value is nondeterministic on each step. If `delaying` has been true on the previous cycle, the previous values of $s_1, ..., s_k$ are returned. Otherwise, their most recent values are returned. However, such an implementation may lead to changes in signal value (e.g., rising and falling edges of Boolean signals) being lost. Hence, the following additional condition restricts the nondeterminism of the module: if `delaying` is true and the value of either of the signals changes between the current and the next cycles, then the next value of `delaying` must be kept true.

To implement a multiple-cycle delay, several delay modules are connected in a chain of length $d_{\max}$. In MODCHK, for visual simplicity, such a chain can be hidden inside a single block. We found that delaying input signals of APUs with $d_{\max} = 3$ and their output signals with $d_{\max} = 6$ leads to considering all possible execution orders of APUs and the ALU in response to changed measurements inputs.

Note that asynchronous behavior can in principle be modeled in NuSMV. However, the solution with the `process` keyword is not compatible with MODCHK, and explicit modeling of asynchrony would involve modification of the code of each module, which is technically difficult to achieve in MODCHK. This is the reason why the approach described above is used.

VII. Case study

Our case study is based on the U.S. EPR Protection System (see Section II-B). The model consist of APU and ALU modules distributed between four redundant divisions. For the application logic, we constructed a fictitious safety function, inspired by the PS function diagrams in [12]. To approach the complexity of detailed design, signal status processing was also added (partly based on publicly available information, partly on invention) within elementary blocks, and also in the shape of special blocks for fault status filtering.

The safety function processing logic for the APUs is shown in Fig. 3 and for the ALU in Fig. 4. For the ALU logic, we added a block type of our own invention (SFV) to substitute the value of faulty control signals from the operator's interface with a default "false".

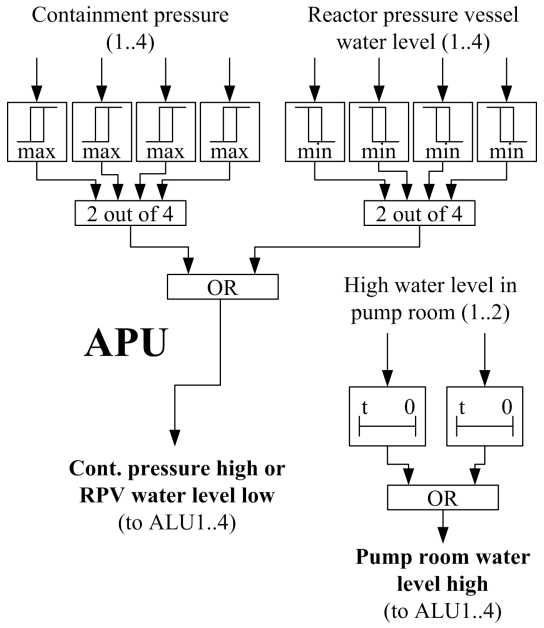

Fig. 3. The processing logic for each of the four redundant APUs.

The function block diagrams were modeled with MODCHK, and functional requirements were translated into LTL, CTL and PSL properties. The number of reachable states in the generated software-only NuSMV model (no failures, communication delay nor asynchrony) is $7.3 \cdot 10^{31}$.

A. Experimental design

We evaluated our modeling approach in four cases, which are distinguished by the absence (F^-) or presence (F^+) of the failure in the model (as described in Section V), and by the absence (D^-) or presence (D^+) of delays (as described in Section VI), i.e., asynchrony modeling. These cases are indicated as F^-D^-, F^+D^-, F^-D^+, and F^+D^+, respectively.

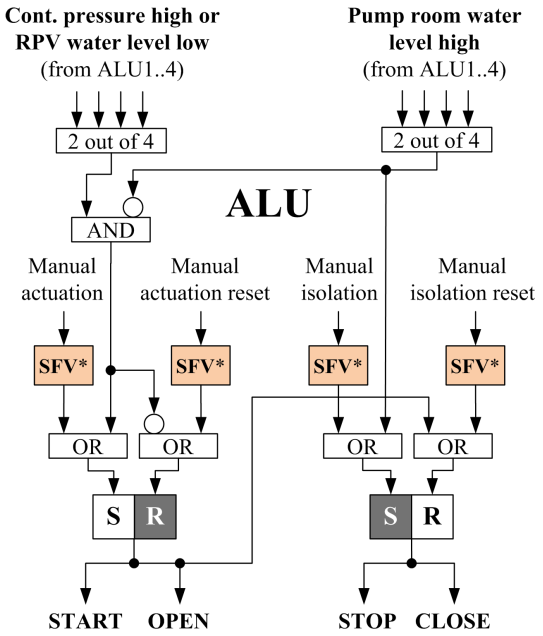

Fig. 4. The processing logic for the ALU.

Technically, we prepared two MODCHK models—with and without failure modeling, both containing placeholders of delay blocks. These models were exported as NuSMV code, wherein the placeholders were replaced with either identity blocks (i.e., the ones which output their inputs), modeling the situation of zero delays, and chains of basic delay blocks to reach $d_{\max} = 3$ and $d_{\max} = 6$ (as described in the end of Section VI).

A list of 13 functional requirements was prepared, including one invariant, 6 request-response properties, 2 absence of spurious actuation properties, and 4 absence of deadlocks[3] properties. The requirements from this list were implemented specifically for each of the four cases. Absence of deadlock requirements were possible to be formulated only in CTL. All other requirements were formulated in both LTL/PSL and CTL, with the exception of the cases with delays, in which the requirements were only formulated in LTL/PSL. Below, as examples, invariant and absence of deadlocks requirements (which are identical regardless of failures and delays) are given in CTL:

1) $\mathbf{AG}\,\neg(\texttt{START} \wedge \texttt{STOP})$;
2) $\mathbf{AG}\,\mathbf{EF}\,\texttt{START}$;
3) $\mathbf{AG}\,\mathbf{EF}\,\texttt{STOP}$;
4) $\mathbf{AG}\,\mathbf{EF}\,\neg\texttt{START}$;
5) $\mathbf{AG}\,\mathbf{EF}\,\neg\texttt{STOP}$.

To check these temporal requirements, three different model checking algorithms were executed: BMC, BDD-based LTL model checking, and BDD-based CTL model checking. BMC was performed with bound $k = 20$. All the experiments were done in Linux Subsystem of Windows 10, on a PC with 2 GHz quad-core CPU (only one core was used to execute NuSMV). NuSMV was run with command line options "`-dynamic -coi -df`".

[3]Technically, these are not deadlocks in the sense of model checking, but situations wherein certain outputs of the ALU become unchangeable.

B. Experimental results

Table I shows the results of model checking runs. The situation without delays was best handled by BDD-based model checking algorithms. Among them, CTL model checking was capable of proving absence of deadlock requirements. As for the case with delays, only BMC was able to check the requirements. Consequently, checking absence of deadlock requirements was impossible in the case with delays.

TABLE I
EXPERIMENTAL RESULTS

Case	Model checking time (s)		
	BMC	LTL	CTL
F^-D^-	98	3	3
F^-D^+	117	—*	—*
F^+D^-	25125	38	7
F^+D^+	138	—*	—*

* Neither of the properties were checked within the time limit of 8 hours

VIII. CONCLUSION AND FUTURE WORK

Previous attempts at introducing hardware failure modes to I&C application logic model checking have resulted in prohibitively complex models and excessive analysis times.

Our experiments with a failure model show that as long as we focus on single failure tolerance, both LTL/PSL and CTL properties can be easily verified with BDD-based model checking. In case of LTL properties, there was a significant, but still practically tolerable increase in the processing times. The failure modeling approach is also easily applied in the graphical tools we use.

However, introducing communication delay and asynchrony to the model meant that BDD-based verification was no longer a viable option. Analysis with BMC took less than 5 minutes, which is still tolerable, but at the expense of coverage of possible scenarios.

Here, we evaluated our modeling approach using a model of a semi-fictitious nuclear reactor protection system, although aiming at the complexity of real-world detailed designs. In the future, the failure modeling approach should be further experimented with on an industrial scale. VTT's customer projects could provide the opportunity. In addition, the tools we use should also be further developed—MODCHK needs to allow the analyst to specify the I&C hardware architecture, and the allocation of the application logic to physical devices. Fault tolerance analysis could then be an automated feature.

For further research, closed-loop modeling is still relevant from the point of view of overall safety. If we shift the focus to plant level interaction of several different systems,

analysis cannot longer be limited to single failures within one isolated system. As issues such as common cause failure become relevant, other abstractions and simplifications are needed. One topic VTT is also working on is using the results from probabilistic safety assessment (PSA) to specify the most critical failure points in the system architecture, and then injecting the failures at those points for application logic model checking.

ACKNOWLEDGMENT

This work has been funded by the Finnish Research Programme on Nuclear Power Plant Safety 2015-2018 (SAFIR 2018), and by the Government of the Russian Federation (Grant 08-08).

We wish to thank Mika Johansson, Kim Wahlström and Nina Lahtinen of STUK for valuable discussions and comments.

REFERENCES

[1] E. M. Clarke, O. Grumberg, and D. Peled, *Model checking*. MIT press, 1999.

[2] A. Pakonen, T. Tahvonen, M. Hartikainen, and M. Pihlanko, "Practical applications of model checking in the Finnish nuclear industry," in *10th International Topical Meeting on Nuclear Plant Instrumentation, Control and Human Machine Interface Technologies (NPIC & HMIT 2017)*. American Nuclear Society, 2017, pp. 1342–1352.

[3] Areva NP. (2012) U.S. EPR Protection System, Technical Report ANP-10309NP, Revision 4. [Online]. Available: https://www.nrc.gov/docs/ML1216/ML121660317.html

[4] M. P. E. Heimdahl, Y. Choi, and M. W. Whalen, "Deviation analysis: A new use of model checking," *Automated Software Engineering*, vol. 12, no. 3, pp. 321–347, Jul 2005.

[5] A. Joshi and M. P. E. Heimdahl, "Model-based safety analysis of Simulink models using SCADE Design Verifier," in *Computer Safety, Reliability, and Security*, R. Winther, B. A. Gran, and G. Dahll, Eds. Springer Berlin Heidelberg, 2005, pp. 122–135.

[6] F. Schneider, S. M. Easterbrook, J. R. Callahan, and G. J. Holzmann, "Validating requirements for fault tolerant systems using model checking," in *IEEE International Symposium on Requirements Engineering (RE '98)*, 1998, pp. 4–13.

[7] M. Zhang, Z. Liu, C. Morisset, and A. P. Ravn, "Design and verification of fault-tolerant components," in *Methods, Models and Tools for Fault Tolerance*, M. Butler, C. Jones, A. Romanovsky, and E. Troubitsyna, Eds. Springer Berlin Heidelberg, 2009, pp. 57–84.

[8] C. Bernardeschi, A. Fantechi, and S. Gnesi, "Model checking fault tolerant systems," *Software Testing, Verification and Reliability*, vol. 12, no. 4, pp. 251–275, Dec 2002.

[9] S. Sharvia and Y. Papadopoulos, "Integrating model checking with HiP-HOPS in model-based safety analysis," *Reliability Engineering & System Safety*, vol. 135, pp. 64–80, 2015.

[10] STUK. (2013) YVL B.1 Safety design of a nuclear power plant. [Online]. Available: https://www.stuklex.fi/en/ohje/YVLB-1

[11] A. Pakonen and K. Björkman, "Model checking as a protective method against spurious actuation of industrial control systems," in *27th European Safety and Reliability Conference (ESREL 2017)*. Taylor & Francis Group, London, UK, 2017, pp. 3189–3196.

[12] Areva NP. (2013) U.S. EPR Final Safety Analysis Report. [Online]. Available: https://www.nrc.gov/reactors/new-reactors/design-cert/epr/reports.html

[13] C. Eisner and D. Fisman, *A Practical Introduction to PSL*. Springer US, 2006.

[14] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV version 2: An opensource tool for symbolic model checking," in *International Conference on Computer-Aided Verification (CAV 2002)*, ser. LNCS, vol. 2404. Springer, 2002.

[15] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L.-J. Hwang, "Symbolic model checking: 10^{20} states and beyond," *Information and Computation*, vol. 98, no. 2, pp. 142–170, 1992.

[16] T. Ovatman, A. Aral, D. Polat, and A. O. Ünver, "An overview of model checking practices on verification of PLC software," *Software & Systems Modeling*, vol. 15, no. 4, pp. 937–960, Oct 2016.

[17] B. F. Adiego, D. Darvas, E. B. Viñuela, J. Tournier, S. Bliudze, J. O. Blech, and V. M. G. Suárez, "Applying model checking to industrial-sized PLC programs," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 6, pp. 1400–1410, 2015.

[18] S. Preuße, H. Lapp, and H. Hanisch, "Closed-loop system modeling, validation, and verification," in *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies Factory Automation (ETFA 2012)*, 2012, pp. 1–8.

[19] I. Buzhinsky, A. Pakonen, and V. Vyatkin, "Explicit-state and symbolic model checking of nuclear I&C systems: A comparison," in *43rd Annual Conference of the IEEE Industrial Electronics Society (IECON 2017)*, 2017, pp. 5439–5446.

[20] A. Pakonen, T. Mätäsniemi, J. Lahtinen, and T. Karhela, "A toolset for model checking of PLC software," in *18th IEEE Conference on Emerging Technologies & Factory Automation (ETFA 2013)*. IEEE, 2013, pp. 1–6.

[21] Rolls-Royce, "Spinline™, A Rolls-Royce modular I&C digital platform dedicated to nuclear safety," Technical sheet, 2012.

[22] J. Lahtinen, "Hardware failure modelling methodology for model checking," VTT, Tech. Rep. VTT-R-00213-14, 2014.

[23] J. Lahtinen, T. Launiainen, K. Heljanko, and J. Ropponen, "Model checking methodology for large systems, faults and asynchronous behaviour. SARANA 2011 work report," VTT, Tech. Rep. VTT Technology 12, 2012.

[24] G. J. Holzmann, "The model checker SPIN," *IEEE Transactions on Software Engineering*, vol. 23, no. 5, pp. 279–295, 1997.

[25] G. Behrmann, A. David, K. G. Larsen, J. Hakansson, P. Petterson, W. Yi, and M. Hendriks, "UPPAAL 4.0," in *3rd International Conference on the Quantitative Evaluation of Systems (QEST 2006)*, Sept 2006, pp. 125–126.

[26] R. Leveugle, "A new approach for early dependability evaluation based on formal property checking and controlled mutations," in *11th IEEE International On-Line Testing Symposium*, July 2005, pp. 260–265.

[27] *Failure modes and effects analysis (FMEA and FMECA)*, IEC Std. 60 812:2018, 2018.

[28] L. Grunske, K. Winter, N. Yatapanage, S. Zafar, and P. A. Lindsay, "Experience with fault injection experiments for FMEA," *Software: Practice and Experience*, vol. 41, no. 11, pp. 1233–1258, Jan 2011.

[29] V. Molnár and I. Majzik, "Model checking-based software-FMEA: Assessment of fault tolerance and error detection mechanisms," *Periodica Polytechnica Electrical Engineering and Computer Science*, vol. 61, no. 2, pp. 132–150, 2017.

[30] H. Aljazzar, M. Fischer, L. Grunske, M. Kuntz, F. Leitner-Fischer, and S. Leue, "Safety analysis of an airbag system using probabilistic FMEA and probabilistic counterexamples," in *6th International Conference on the Quantitative Evaluation of Systems (QEST 2009)*, Sept 2009, pp. 299–308.