POLITECNICO DI TORINO
Repository ISTITUZIONALE

Validating the Design of CPS: Interfacing Simulations of Multi-Physics Components and Software with Contract-Based Monitoring

Original
Validating the Design of CPS: Interfacing Simulations of Multi-Physics Components and Software with Contract-Based Monitoring / Bruns, F., Tosoni, F., Mehlhop, S., Rauh, A., Vinco, S., Walter, J., Oppenheimer, F., Fummi, F.. - ELETTRONICO. - (2025), pp. 35-40. (International Conference on Methods and Models in Automation and Robotics, MMAR Miedzyzdroje (POL) 26-29 August 2025) [10.1109/MMAR65820.2025.11150819].

Availability:
This version is available at: 11583/3011009 since: 2026-05-18T16:09:17Z

Publisher:
IEEE

Published
DOI:10.1109/MMAR65820.2025.11150819

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©2025 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

18 November 2026

Validating the Design of CPS: Interfacing Simulations of Multi-Physics Components and Software with Contract-Based Monitoring

Friederike Bruns*, Francesco Tosoni†, Sven Mehlhop‡, Andreas Rauh*,
Sara Vinco§, Jörg Walter‡, Frank Oppenheimer‡, Franco Fummi†
*Dept. of Computing Science, Carl von Ossietzky Universität Oldenburg, Germany, {friederike.bruns, andreas.rauh}@uol.de
†Dept. of Engineering IM, University of Verona, Italy, {francesco.tosoni, franco.fummi}@univr.it
‡OFFIS e.V. Institut für Informatik, Oldenburg, Germany, {sven.mehlhop, joerg.walter, frank.oppenheimer}@offis.de
§Dept. of Control and Computer Engineering, Politecnico di Torino, Italy, {sara.vinco}@polito.it

***Abstract*—Ensuring fault tolerance in Cyber-Physical Systems (CPSs) is challenging due to their complexity and stringent safety requirements. Modern fault-tolerant approaches guarantee fault detection, isolation, and mitigation, but lack systematic approaches to prove their effectiveness and correctness. This paper presents a simulation framework integrating fault injection and contract-based monitoring to validate fault tolerance under diverse conditions. Unlike nominal behavior-based methods, it refines contract specifications through fault-driven scenarios, defining acceptable fault severity and enhancing trust in detection mechanisms. This approach enables early fault detection and precise assessment of critical components by supporting continuous monitoring and allowing prompt corrective actions, improving fault management in dynamic environments. A proof-of-concept implementation demonstrates the framework's effectiveness in assessing fault impacts both in multi-physics components and their controller modules, highlighting its potential to enhance the reliability and resilience of complex CPSs.**

***Index Terms*—Co-Simulation, Fault Injection, Assumption-Guarantee Contracts**

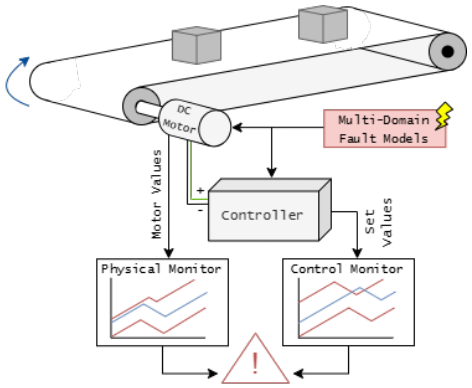


Fig. 1: Co-simulation of software and physical components for multi-domain fault detection in CPS.

I. Introduction

Ensuring fault tolerance in CPSs is especially crucial in environments characterized by complex dynamics and stringent safety requirements. Traditional fault-tolerant control strategies often rely on dedicated algorithms for Fault Detection and Isolation (FDI) [1]. However, validating the effectiveness of these algorithms or understanding a system's behavior in the absence of specific fault-tolerant measures necessitates systematic and comprehensive simulation and testing.

A significant challenge in FDI and control lies in ensuring that the correct components of a system are monitored to accurately identify anomalous behaviors: monitoring the wrong components might lead to a shadowing of faults like component wear by control algorithms, eventually resulting in unintended effects such as increased set values, ultimately causing violations of input or state constraints.

This paper presents a novel method that interfaces simulation of control software with detailed physical simulation, both enhanced with injected behavioral faults and contract-based monitors, to evaluate fault tolerance in CPSs. Fig. 1 exemplifies a use case in which a control unit regulates the speed and position of a conveyor belt driven by a DC motor: faults can be injected into both the DC motor and the controller, representing the classes of physical and digital faults. Our approach enhances system reliability and robustness by providing a solid foundation for FDI and fault mitigation, such as continuous monitoring and timely corrective actions. It helps developers assess how long a system can maintain acceptable performance under various fault conditions. This is achieved by integrating contract-based monitoring for all subsystems, refining them through fault injection, and by modeling faulty behavior to improve monitor accuracy and robustness. This systematic, fault-driven approach enhances CPSs design, offering deeper insights into system robustness and fault-tolerant strategies. Thus, this work focuses on:

1) **Consistent and comprehensive** assessment, by maintaining a uniform level of detail across control software and physical simulations for a wide range of faults.
2) **Early fault detection** and accurate specification of requirements, with a focuses on **single faults** and the extensibility for multiple simultaneous faults.
3) Multiple **systematically designed** contract-based mon-

itors to observe set values and measured values to effectively detect potential faults.

4) **Formal representation** of monitors grounded in assume-guarantee specifications.

Section II covers related work and Section III presents the concept for the simulation enriched with fault categorization, fault injection, and contract-based monitoring. Section IV describes the proof-of-concept implementation, followed by a use case study and discussion of the evaluation scenarios in Section V. The final Section VI summarizes the work and provides an outlook on future research.

II. Related Work on Fault Simulation & Analysis

FDI and tolerance in CPSs have been extensively researched across domains [2]. Nardi et al. [3] emphasize improving FDI through comprehensive modeling and simulation, while the IEEE P2427 standard [4] focuses on analog defect modeling. Balaban et al. [5] and Uder et al. [6] highlight fault analysis in aerospace environments. Simulation plays a critical role in fault detection. Kolesnikov et al. [7] simulate fault behaviors in vehicle dynamics, and Gundermann et al. [8] provide a fault library for Modelica. Pan et al. [9] emphasize the complementary roles of SystemC and Matlab for system validation and development. Additionally, the fault detection capabilities in languages like Verilog-AMS [10], [11] and SystemC-AMS [12] further underline the diverse ecosystem of simulation frameworks available for CPS fault tolerance research. Cabral et al. [13] enhance FDI via co-simulation for IEC 61499 applications, while Feiler et al. [14] and Mehlhop et al. [15] propose early FDI methods for distributed systems. Bouhadiba et al. [16] ensure alignment with system requirements using formal contracts in SystemC.

A design based on assumption-guarantee contracts together with formal specifications has gained significant interest in CPSs fault detection, as highlighted in [17]–[20], also aligning with robustness and predictive maintenance strategies. Nuzzo et al. [21] focus on enhancing robustness for analog systems, while Zonta et al. [22] review predictive maintenance within the context of Industry 4.0. Recent research also highlights data-driven FDI approaches using machine learning [23], [24], which eliminate the need for system modeling by leveraging machinery operation datasets. However, obtaining datasets, especially those with faulty data traces, remains challenging due to limited availability and company confidentiality.

Our work integrates contract-based monitoring across digital and analog subsystems during design and monitoring phases, using fault injection to assess FDI capabilities of the introduced monitors and to iteratively refine the CPS design. Unlike traditional methods, we incorporate faulty behaviors to enhance detection accuracy and robustness, offering a comprehensive strategy for FDI and mitigation in CPS.

III. Hardware & Physical Components Simulation with Fault Injection and Monitoring

This section presents the proposed framework for modeling control software and physical components of a CPS, incorporating fault injection, and specifying contract-based monitors for fault detection during simulation, cf. Fig. 2.

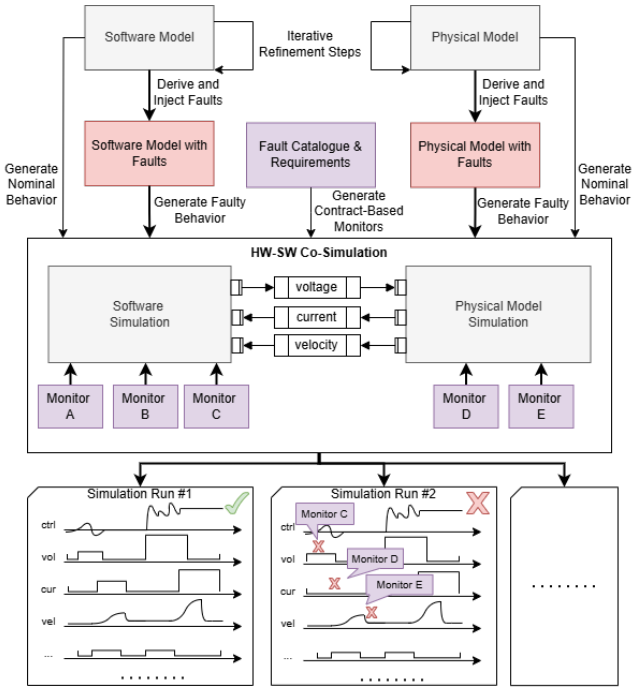


Fig. 2: Modeling software and hardware of a CPS with an injection of faults and contract-based monitors for FDI.

The design of a CPS typically involves iterative refinement steps that address the specific requirements of both control software (left) and physical elements (right). The modular nature of contracts enables testing and verification of individual sub-systems (monitors A to E), which can then be combined through hierarchical contracts to form a reliable overall system. Through this process, a specification of the nominal behavior of the system is developed. We leverage the contract-based methodology to design monitors that can be simulated alongside both the software and physical models, to allow FDI by integrating monitoring mechanisms within the simulation. Each simulation run allows for an evaluation of the observed behavior and an analysis of the FDI capability of the monitors. In Fig. 2, simulation run $\#1$ exemplifies a simulation with no faults detected, while simulation run $\#2$ shows the activation of three monitors on both the software and the physical parts, as an effect to detected faults.

In the following we discuss key concepts related to fault categorization and injection, simulation of software and physical parts' interactions, and the design of contract-based monitors.

A. Control & Physical Components: Models & Simulation

A CPS is subject to numerous requirements, influencing the choice of model for each components. For the physical part, the dynamics are commonly described with Differential Algebraic Equations (DAEs). On the software side, developers have access to a variety of modeling languages to meet system

requirements and ensure functionality. Common choices include SysML for system-level design and specification, as well as standards like IEC 61131 or IEC 61499 [25], widely used for designing and programming industrial automation systems.

Various simulation approaches are essential to validate CPSs models, ensuring they meet performance, safety, and reliability standards [26]. Model-in-the-Loop (MiL) and Software-in-the-Loop (SiL) simulations enable early testing without physical hardware. Hardware-in-the-Loop (HiL) simulations integrate real hardware with simulated environments to test interactions in real-world scenarios. Co-simulation allows simultaneous testing of physical elements (e.g., circuit dynamics) and software (e.g., control logic) to ensure seamless integration. Combined with formal methods (e.g., model checking and runtime verification), simulation ensures comprehensive validation, reducing risks and improving reliability before deployment.

The methodology presented in this paper is based primarily on system simulation. From the languages outlined in Section II, SystemC with its AMS extension is chosen as it covers a wide range of domains in a single environment (e.g., electrical and mechanical components) [27]. In detail, SystemC-AMS supports TDF for static scheduled discrete time processes (e.g., digital hardware or control algorithms), LSF for DAE models, and ELN for Electrical Linear Networks. These different types of models can be used simultaneously in the same simulation due to the seamless interaction through the underlying SystemC simulation kernel.

B. Fault Categorization & Fault Injection

Automated systems are inherently vulnerable to faults, which can arise from defects in sensors, actuators, the underlying processes, or the control algorithms themselves, cf. [2], [24]. In closed-loop control systems, faults can gradually worsen, leading to performance degradation and eventual system failure. These faults may remain hidden until a critical failure point is reached, potentially causing sudden halting of production or severe disruptions to operation. Fault injection and simulation are essential to overcome such a late detection. In CPSs, both control and physical components must be validated. Fault injection tests system behavior under various scenarios, identifying the most critical conditions. For the physical part, faults are simulated by modifying the structure and/or parameters of system models. In software, faults can be injected into the model by adding code to alter data transmission or values, allowing for an analysis of their impact on the system performance.

C. Contract-Based Monitors

The underlying methodology for contract-based monitors is Contract-Based Design (CBD) [28], which enhances modularity and reusability. Each system component is described by a contract composed of assumptions and guarantees. The assumptions define expected conditions from the environment, such as specific parameter values, while the guarantees describe the component's behavior if the assumptions hold. This approach allows early detection of potential design issues and improves overall system reliability. In combination with fault injection, we propose to use contract-based monitors as a possibility to determine the root cause of an error that could occur during runtime. Thereby, establishing the basis for fault isolation and estimation. Contracts can be expressed in different formal languages, such as MTSL [29] or StSTL [30]. For this work, we chose the concept of Time-Sensitive Behavioral Contracts (TSBCs) [31]. In TSBC monitoring, system behavior is defined by time (t) and data (d) variables. A contract is expressed as $t \in [t_1; t_2]\ ?\ d \in [d_1; d_2]$, which checks if a specific behavior occurs within a given time interval. If the time condition is met (assumption), the corresponding data condition must also hold (guarantee). Any deviation from this specification is flagged as a fault. The required parameters for contract-based monitors can be specified during design time or derived at runtime as functions of reference and corresponding control signals. By synchronizing with system events, TSBC monitors can assess system properties in real-time. If a contract violation occurs, monitors report the fault and determine whether to continue verification (recoverable monitoring) or halt further checks until the next request (unrecoverable monitoring). Depending on the fault severity, adjustments can be made to the simulation model or contract specifications. This could involve refining system behavior through additional simulations, model adjustments, or modifying the contract to more precisely specify permissible behavior. Integrating multiple monitors enhances the precision of FDI but also increases system overhead. This approach is best reserved for critical system components where high monitoring accuracy is essential. In contrast, relaxing contract specifications should be approached with caution, as it may mask potential issues rather than improving system reliability.

IV. Proof-of-Concept Implementation

This section presents the simulation framework used as a proof-of-concept, demonstrated through a DC motor example as a device that is omnipresent in industrial machinery. We introduce the physical model with its injected faults, and contract-based monitors, followed by the control software, its fault scenarios, and the corresponding contract-based monitors.

A. DC Motor Model

In our illustrating scenario, the DC motor serves as the main drive of a conveyor belt. The motor model, described by Eqs. (1) and (2), is coded in the SystemC-AMS language, using models for its electrical and mechanical behavior:

$$V = K_E \cdot \omega + R_M \cdot i + L_M \cdot \frac{\mathrm{d}i}{\mathrm{d}t} \tag{1}$$

$$\tau = K_T \cdot i - B \cdot \omega - J \cdot \frac{\mathrm{d}\omega}{\mathrm{d}t} \tag{2}$$

The parameters of the used motor are listed in Table I. Acc. to Eqs. (1) and (2), the motor produces shaft rotation as a result of an input voltage, with an acceleration for positive inputs and a deceleration for negative ones. The DC motor behavior could be affected by various faults in both its

TABLE I: DC motor parameters.

Variable	Name	Value	SI Unit
Input voltage	V	5	V
Back-emf constant	K_E	0.383	V/(rad/s)
Torque coefficient	K_T	383	mNm/A
Armature resistance	R_M	1.5	Ω
Armature inductance	L_M	1.5	mH
Motor inertia	J	0.0021	$\mathrm{kg} \cdot \mathrm{m}^2$
Motor friction	B	0.0088	$\mathrm{Nms/rad}$

TABLE II: Overview of example contract specifications for the velocity value observable at software and physical component.

Contract ID	**Contract Specification**
C_SW_Vel1	$t \in [0.00; 0.15]\ ?\ d \in [-1.78; 19.54]$
C_HW_Vel1	$t \in [0.00; 0.15]\ ?\ d \in [-1.84; 20.23]$
C_SW_Vel2	$t \in [0.15; 1.1]\ ?\ d \in [11.69; 12.18]$
C_HW_Vel2	$t \in [0.15; 1.1]\ ?\ d \in [11.72; 12.19]$

electrical and mechanical submodels. In the results presented in the following sections, we restrict ourselves to two faults to show the effectiveness of fault injection: (i) an electrical fault resulting from an increased resistance to of 20.5Ω; (ii) a mechanical fault leading to an increase of the velocity-proportional friction to $50\mathrm{Nms/rad}$.

To detect faulty behavior, we employ three monitors: one for the electric current in the armature circuit, one for the angular speed of the shaft, and one for the applied voltage. These monitors are based on the previously introduced TSBCs and are specified in Table II, where the motor contracts have the identifier `hw`, while the software contracts are marked by `sw`.

B. Control Software Model

Due to the wide use of motors in Industrial Cyber-Physical Systems (ICPSs), IEC 61499 was chosen as modeling language for specifying the control software implementation, cf. Fig. 3.

The control software consists of five Function Blocks (FBs) and represents a standard input–process–output model with the FB `E_Cycle` as clock generator to trigger execution every millisecond. The interfaces between software and physical are covered within the FBs `ReadSensor` and `SetMotor` which convert analog to digital signals and vice versa. The `ReadSensor` FB gets the values `velocity`, `current`, and `position` as an input for the software model. The output of `SetMotor` provides the input `voltage` to the DC motor. Based on the angular position, `DriveControl` may specify different reference values for the Proportional–integral–derivative controller (PID) FB. For simplicity, the desired speed profile is specified as an acceleration to a velocity of $12\,\mathrm{rad/s}$, slowing down to $3\,\mathrm{rad/s}$ after a given time span, before the `velocity` set point is set to zero. The PID is implemented based on [25] and consists of a proportional (`PID_Cal`), integral (`IntegralReal`), and derivative (`DerivativeReal`) part. The PID can be tuned with respect to the following inputs: MODE, a boolean value that controls whether the set point is configured automatically or manually (true = auto or false = manual); ManOut (manual set output); the process value and set point; KP (proportional gain); KI (integration constant 1/sec); TD (derivative time, sec); and lastly, Cycle, used as the discretization step size of the integral and time derivative.

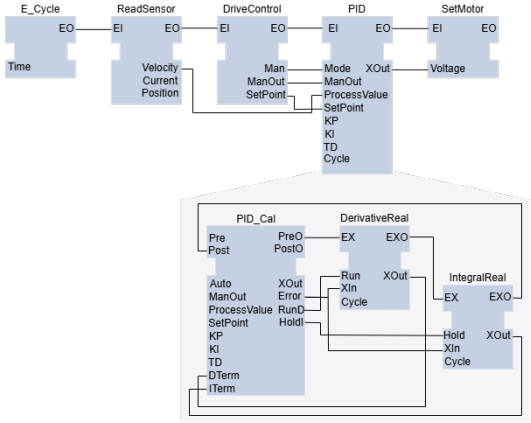


Fig. 3: Software model using a PID-controller in IEC 61499.

The simulator for the software model is built in SystemC and mimics the execution behavior of 4diac FORTE [32] as a specific implementation of IEC 61499. The control software is created within the open-source 4diac IDE. It is then transformed into SystemC bya model-to-model transformation.

Two different faults are prepared to be injected into the software component. For the first, the last input value will be used with a probability of 6% instead of the new values for a random period between 1-10 cycles of $10\,\mathrm{ms}$. The second one corresponds to a random output perturbation of FB `SetMotor` by $\pm - 10\,\%$. While the values of the first fault can be monitored, those of the second one can not.

Similarly to the motor simulation, runtime monitors for the set value in `SetMotor` FB or the measured `velocity` in the `ReadSensor` FB are implemented in the software simulation. These monitors observe contracts based on TSBC. For examples, see Table II.

V. Results

Fig. 4 summarizes simulations in four plots for the observed voltage, velocity, current, and the integral value of the PID, paired with their corresponding monitor values. Voltage, velocity, and current are shown for both the physical (red) and software (blue) sides, while the integral value is only available in the software model. Monitors respond to valid behavior, returning *false* upon detecting violations. Admissible ranges are shaded in light grey for software and dark grey for physical quantities. The plots are divided into five scenarios, each with a 3-second simulation window.

The first scenario illustrates the model's *nominal behavior* using a PID controller with initial parameters set by the developer, potentially based on arbitrary but seemingly adequate choices. White Gaussian noise is added within the `ReadSensor` FB to provide more realistic data. During nominal operation, the system exhibits a stable voltage, velocity, and current, with monitors consistently indicating *true* for all quantities. The PID controller ensures steady-state accuracy.

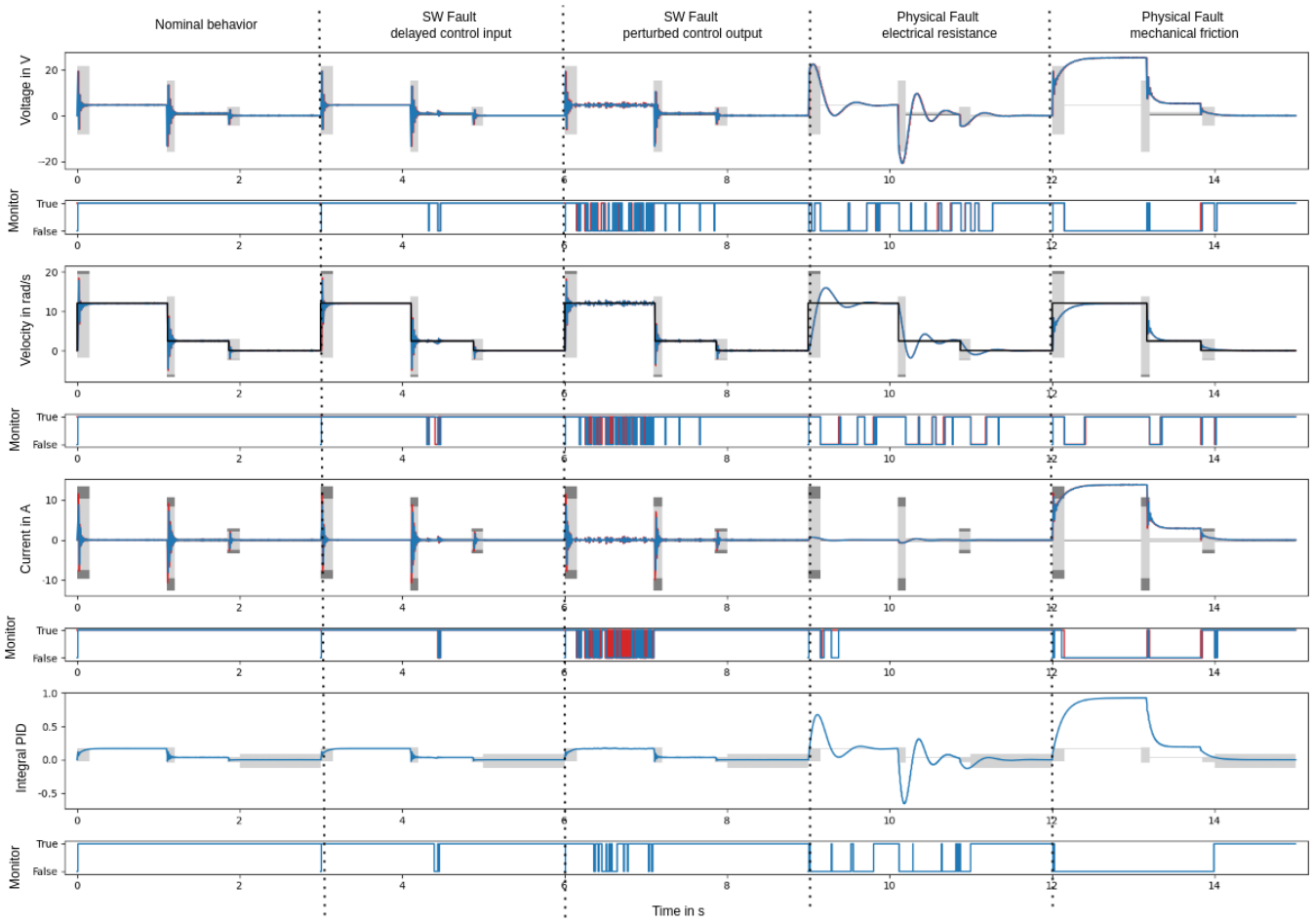


Fig. 4: Trace of the physical model (red signals) and SW (blue signals) values and the corresponding monitor outputs. Valid intervals are highlighted within the corresponding plots for software (light grey) and physical side (dark grey).

However, variations of the parameters of the DC motor component without re-tuning the PID parameters could lead to inefficiencies. To mitigate this risk, well-known automatic PID tuning mechanisms could be integrated in the control block.

In the *first software fault scenario*, a delayed control input disrupts the system, causing transient deviations in voltage and velocity, which the monitors flag as faults. At first sight, the current appears unaffected. However, this statement is only true as long as the introduced delays are not shorter than the system's time constants, which otherwise serve as a natural low-pass filter (at least partially) suppressing this effect. The *second software fault*, involving a perturbed control output, has a more significant influence. Voltage, velocity, and current exhibit fluctuations, and the monitors detect continuous violations. The PID controller's integral value also increases, showing the system's inability to maintain proper control.

Next, with the introduction of a *physical fault affecting the electrical resistance*, the system experiences oscillations and loses its rapid settling behavior. An abnormal behavior of the voltage, velocity, and current is detected by the monitoring system repeatedly. However, the current is way smaller than expected, which even goes unnoticed based on the monitor specification. This could be solved with a more precise monitor specification. The PID controller shows a flagged deviation from its expected trajectory, reflecting the increased effort of maintaining control. In the final phase, a *mechanical friction fault* affects the system's velocity and current. As friction increases, velocity drops, and current spikes to compensate for the load. The monitors accurately detect these faults, while voltage remains stable, highlighting the need for multi-domain monitoring.

The monitoring system efficiently detects both software and physical faults with precise timing. Software faults with an intermittent behavior cause transient issues, while physical faults lead to more extended disruptions. Multi-parameter monitoring is crucial for reliable FDI, especially as control algorithms may mask physical issues. Note that scalability may become a concern as the complexity of control algorithms and the number of hardware components increase. Automating monitor generation from contract specifications can facilitate integration. Our approach aids developers in efficiently and reliably placing monitors to detect a variety of potential faults.

VI. Conclusion & Future Work

In conclusion, this paper presents a novel approach for integrated software-physical simulation with injected faults and contract-based monitors for early fault detection in CPSs. The results demonstrate effective identification of both software and physical faults across key system parameters, ensuring early detection and robust fault isolation. This helps developers in identifying which monitors are relevant for their specific system and assess which of the simulated faults are at higher risk to lead to potential system failure. While the approach shows strong potential, scalability needs to be considered for when system complexity grows. This is considered within the underlying contract-based design methodology supporting a modular, reusable, and updatable CPS design.

Regarding future directions, our framework supports the comparison of different fault estimation and fault-tolerant control procedures. This could be validated systematically through case studies comprising an exhaustive list of faults in complex, interconnected systems together with their associated software components. Moreover, the detection of multiple and common cause faults will be analyzed by an integration of appropriate fault isolation schemes in the monitors. They especially have to deal with slow degradation phenomena that are relevant in the field of predictive maintenance. Additionally, the methodology could be expanded by integrating advanced software components for FDI, replacing simple threshold monitors with more sophisticated fault estimation methods.

References

[1] H. Noura, D. Theilliol, J.-C. Ponsart, and A. Chamseddine, *Fault-Tolerant Control Systems: Design and Practical Applications*. Springer Science & Business Media, 08 2009.

[2] M. Blanke, M. Kinnaert, J. Lunze, and M. Staroswiecki, *Diagnosis and Fault-Tolerant Control*, 2nd ed. Springer Berlin, Heidelberg, 01 2006.

[3] A. Nardi, S. Camdzic, A. Armato, and F. Lertora, "Design-for-safety for automotive ic design: Challenges and opportunities," *2019 IEEE Custom Integrated Circuits Conference (CICC)*, 2019.

[4] "IEEE-SA standards board P2427/D0.13 draft standard for analog defect modeling and coverage," 2019.

[5] E. Balaban, P. Bansal, P. Stoelting, A. Saxena, K. F. Goebel, and S. Curran, "A diagnostic approach for electro-mechanical actuators in aerospace systems," in *2009 IEEE Aerospace conference*, 2009.

[6] S. Uder, R. Stone, Associate, and I. Tumer, "Failure analysis in subsystem design for space missions," in *ASME Design Theory and Methodology Conference*, vol. 3, 01 2004.

[7] A. Kolesnikov, D. Tretsiak, and M. Cameron, "Systematic simulation of fault behavior by analysis of vehicle dynamics," in *Proceedings of the 13th International Modelica Conference, Regensburg, Germany,*, 02 2019, pp. 451–460.

[8] J. Gundermann, A. Kolesnikov, M. Cameron, and T. Blochwitz, "The fault library - a new modelica library allows for the systematic simulation of non-nominal system behavior," *Proceedings of the 2nd Japanese Modelica Conference Tokyo, Japan*, 2019.

[9] X. Pan, C. Zivkovic, and C. Grimm, "Virtual Prototyping of Heterogeneous Automotive Applications: Matlab, SystemC, or both?" in *24th Asia and South Pacific Design Automation Conference, Tokyo, Japan*, 2019, pp. 544–549.

[10] A. Ballo, M. Bottaro, A. D. Grasso, and G. Palumbo, "Regulated Charge Pumps: A Comparative Study by Means of Verilog-AMS," *Electronics*, vol. 9, no. 6, p. 998, 2020.

[11] F. Tosoni, N. Dall'Ora, E. Fraccaroli, S. Vinco, and F. Fummi, "Multidomain Fault Models Covering the Analog Side of a Smart or Cyber–Physical System," *IEEE Transactions on Computers*, vol. 73, no. 3, pp. 829–841, 2024.

[12] B. Vernay, A. Krust, G. Schröpfer, F. Pêcheux, and M.-M. Louerat, "SystemC-AMS Simulation of a Biaxial Accelerometer based on MEMS Model Order Reduction," in *Symposium on Design, Test, Integration and Packaging of MEMS/MOEMS (DTIP), Montpellier, France*. IEEE, 2015.

[13] J. Cabral, M. Wenger, and A. Zoitl, "Enable co-simulation for industrial automation by an fmu exporter for IEC 61499 models," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1. IEEE, 2018, pp. 449–455.

[14] P. H. Feiler, J. Hansson, D. de Niz, and L. Wrage, "System architecture virtual integration: An industrial case study," *Software Engineering Institute Technical Report, CMU/SEI-2009-TR-017*, 2009.

[15] S. Mehlhop and J. Walter, "Model-Aware Simulation of IEC 61499 Designs," in *27th IEEE Int. Conf. Emerging Technologies and Factory Automation (ETFA), Stuttgart, Germany*, 2022.

[16] T. Bouhadiba, F. Maraninchi, and G. Funchal, "Formal and executable contracts for transaction-level modeling in systemc," in *Proceedings of the Seventh ACM International Conference on Embedded Software*, ser. EMSOFT '09, New York, NY, USA, 2009, p. 97–106.

[17] B. Meyer, "Applying 'design by contract'," *Computer*, vol. 25, no. 10, 1992.

[18] W. Damm, H. Hungar, B. Josko, T. Peikenkamp, and I. Stierand, "Using contract-based component specifications for virtual integration testing and architecture design," in *2011 Design, Automation & Test in Europe*, 2011.

[19] A. Cimatti and S. Tonetta, "Contracts-refinement proof system for component-based embedded systems," *Sci. Comput. Program.*, vol. 97, no. P3, p. 333–348, Jan 2015.

[20] V. P. Ivannikov, A. S. Kamkin, A. S. Kossatchev, V. V. Kuliamin, and A. K. Petrenko, "The Use of Contract Specifications for Representing Requirements and for Functional Testing of Hardware Models," in *Programming and Computer Software*, vol. 33, 2007.

[21] P. Nuzzo and A. Sangiovanni-Vincentelli, "Robustness in analog systems: Design techniques, methodologies and tools," in *2011 6th IEEE International Symposium on Industrial and Embedded Systems*, 2011, pp. 194–203.

[22] T. Zonta, C. A. Da Costa, R. da Rosa Righi, M. J. de Lima, E. S. da Trindade, and G. P. Li, "Predictive Maintenance in the Industry 4.0: A Systematic Literature Review," *Computers & Industrial Engineering*, vol. 150, p. 106889, 2020.

[23] Y.-J. Park, S.-K. S. Fan, and C.-Y. Hsu, "A Review on Fault Detection and Process Diagnostics in Industrial Processes," *Processes*, vol. 8, no. 9, 2020.

[24] A. Abid, M. T. Khan, and J. Iqbal, "A review on fault detection and diagnosis techniques: basics and beyond," *Artificial Intelligence Review*, vol. 54, no. 5, pp. 3639–3664, 2021.

[25] A. Zoitl and R. Lewis, *Modelling Control Systems Using IEC 61499*, 2nd ed. Institute of Electrical and Electronics Engineers (IEEE), 2014.

[26] J. Sini, M. Violante, and R. Dessì, "Computer-aided design of multi-agent cyber-physical systems," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2018, pp. 677–684.

[27] E. Fraccaroli and S. Vinco, "Modeling cyber-physical production systems with systemc-ams," *IEEE Transactions on Computers*, vol. 72, no. 7, pp. 2039–2051, 2023.

[28] A. Sangiovanni-Vincentelli, W. Damm, and R. Passerone, "Taming Dr. Frankenstein: Contract-Based Design for Cyber-Physical Systems," *European Journal of Control*, vol. 18, no. 3, pp. 217–238, 2012.

[29] E. Böde, W. Damm, G. Ehmen, M. Fränzle, K. Grüttner, P. Ittershagen, B. Josko, B. Koopmann, F. Poppen, M. Siegel, and I. Stierand, "MULTIC-Tooling," in *FAT-Schriftenreihe 316*, 2019.

[30] C. Oh, M. Lora, and P. Nuzzo, "Quantitative verification and design space exploration under uncertainty with parametric stochastic contracts," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '22, New York, NY, USA, 2022.

[31] D. Do Tran, J. Walter, K. Grüttner, and F. Oppenheimer, "Towards time-sensitive behavioral contract monitors for iec 61499 function blocks," in *2020 IEEE Conference on Industrial Cyberphysical Systems (ICPS)*, vol. 1, 2020, pp. 27–34.

[32] Eclipse Foundation, "What is Eclipse 4diac," https://eclipse.dev/4diac/, 2024.