Validity Frame Concept as Effort-Cutting Technique within the Verification and Validation of Complex Cyber-Physical Systems

Bert Van Acker
Bentley James Oakes
Mehrdad Moradi
Paul Demeulenaere
Joachim Denil
bert.vanacker@uantwerpen.be
bentley.oakes@uantwerpen.be
mehrdad.moradi@uantwerpen.be
paul.demeulenaere@uantwerpen.be
joachim.denil@uantwerpen.be
University of Antwerp and Flanders Make vzw, Belgium

ABSTRACT

The increasing performance demands and certification needs of complex cyber-physical systems (CPS) raise the complexity of the engineering process, not only within the development phase, but also in the Verification and Validation (V&V) phase. A proven technique to handle the complexity of CPSs is Model-Based Design (MBD). Nevertheless, the verification and validation of complex CPSs is still an exhaustive process and the usability of the models to front-load V&V activities heavily depends on the knowledge of the models and the correctness of the conducted virtual experiments. In this paper, we explore how the effort (and cost) of the V&V phase of the engineering process of complex CPSs can be reduced by enhancing the knowledge about the system components, and explicitly capturing it within their corresponding validity frame. This effort reduction originates from exploiting the captured system knowledge to generate efficient V&V processes and by automating activities at different model life stages, such as the setup and execution of boundary-value or fault-injection tests. This will be discussed in the context of a complex CPS: a safety-critical adaptive cruise control system.

CCS CONCEPTS

• **Computer systems organization** → **Embedded and cyber-physical systems**; • **Computing methodologies** → **Model verification and validation**.

MODELS '20 Companion, October 18–23, 2020, Virtual Event, Canada

ACM ISBN 978-1-4503-8135-2/20/10…$15.00
https://doi.org/10.1145/3417990.3419226

ACM Reference Format:
Bert Van Acker, Bentley James Oakes, Mehrdad Moradi, Paul Demeulenaere, and Joachim Denil. 2020. Validity Frame Concept as Effort-Cutting Technique within the Verification and Validation of Complex Cyber-Physical Systems. In *ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems (MODELS '20 Companion), October 18–23, 2020, Virtual Event, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3417990.3419226

1 INTRODUCTION

In the development of software intensive systems such as the avionics and automotive domains, engineers need to cope with highly complex devices composed of different interacting and deeply intertwined components [10]. Development of these cyber-physical systems (CPS) is becoming increasingly complex, caused by (i) the synergistic interaction between software and physical elements and (ii) the vast demand for improved performance, e.g. faster and safer control, autonomous driving, or reduced energy consumption. A well-proven design technique for the design of CPSs is **Model-Based Design**. Model-Based Design (MBD) employs (mathematical) modeling to *design*, *analyze*, *verify* and *validate* dynamic and complex systems [3]. More specifically, MBD uses models to represent a system's elements and their relationships, where models serve as input and output at all stages of system development, from *conceptual design phase* and continuing throughout development and later life cycle phases [7]. The following activities are identified in the typical design process of CPSs:

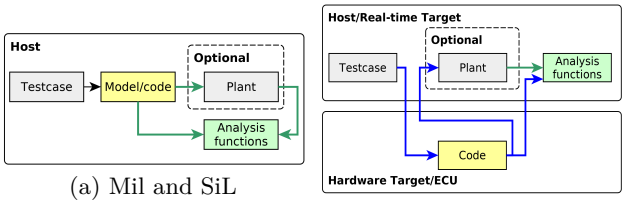
- **Concept definition:** The system-under-design and its influencing environment is *interpreted*, *scoped*, and the *expected behavior* is defined, which will be used to satisfy its goal or purpose.
- **Design:** Implementation of the system-under-design by the engineer(s)
- **Integration:** Integration of the implemented system-under-design

- **Validation:** The process of assessing how well the behavior of the system-under-design is emulated by its virtual counterpart.
- **Verification:** The process of assessing the implemented behavior of the system-under-design against the defined expected behavior. The expected behavior can e.g. be defined using (system) requirements.

Problem statement. There exists a tight coupling between the Design and Integration activities of a design process and the Verification and Validation (V&V) activities. As the requirements for a system-under-design become more complex, it becomes more difficult to design the system and perform the (V&V) activities. Validation is a key activity within a MBD process, as it enables the engineer to determine the **model fidelity**, the degree to which a model faithfully represents the source system counterpart [11, 26]. This is inextricably connected to the *properties-of-interest* of the source system which are encoded in the virtual model. The model fidelity is dependent on numerous influencing factors, ranging from the implementation details to the execution environment in which the model is used. An example is a reduced-order model compared to a high-fidelity model, or whether the model is simulated using a continuous or a discrete solver. The transition from the conventional software development (via hand-coding) to model-based development methods helps abstract away unnecessary details in a manner that increases the potential for easy validation and verification. The model-based approach consists of four elements [17]: (i) models as executable specification, used as common understanding for different stakeholders, such as managers, control and embedded engineers, etc. (ii) design with simulation, validating and verifying the model behavior in a (semi-) virtual environment (iii) implementation through code generation, ECU code generated from the models and (iv) continuous test and verification. More specifically, some V&V activities can be **front-loaded** in the design process, identifying gaps and errors in the system specification in a very early stage of the development. For example, the authors in [16] used model-based testing which utilized front-loaded knowledge about hardware and software. These V&V activities can be carried out at the different stages of the system life cycle:

- *Model level or model-in-the-loop (MiL)*: Validation of the models behavior within a virtual simulated environment.
- *Software level or software-in-the-loop (SiL)*: Validation of ECU code, generated from the model, within a virtual simulated environment.
- *Processor level or or processor-in-the-loop (PiL)*: Validation of the deployed code onto the target hardware, within a virtual simulated environment.
- *Hardware level or or hardware-in-the-loop (HiL)*: Validation of the deployed code onto the target hardware, interacting with the real-world via real-time simulated sensor, actuator and response behavior.

The key difference between the different stages is shown in Figure 1. This shows a conceptual overview of the interplay between virtual and/or physical simulation elements for the aforementioned stages. The model or code under study is presented in yellow, while the analysis functions are green. The virtual links between the simulation elements are shown in green and the physical links in blue.



(a) Mil and SiL (b) PiL and HiL

Figure 1: Conceptual view on different stages

Contribution. This work concerns: (i) capturing knowledge about complex CPSs (and their environment) with one or more models-of-the-physics and (ii) exploiting this knowledge to reduce the effort of the V&V process. We propose the validity frame (VF) concept to capture this knowledge, as VFs enable reasoning about the validity of components by *explicitly capturing everything that alters the validity of models* within their corresponding VF. Section 4 describes the objective, scope, and structure of VFs, which include information such as the solver used for a model and the testing procedure to ensure that the solver produces a correct result.

Therefore, the concrete contribution of the paper is detailing how data and processes contained within the VFs can be exploited to reduce the complexity and effort of the V&V process. That is, how to store and use information for: (i) employing processes supporting V&V activities, (ii) *generating*, *customizing*, and *optimizing* V&V activities, and (iii) *coordinating* and *automating* V&V activities.

Paper Organization. Section 2 presents an illustrative example. Section 3 gives an overview of the related work. Section 4 discusses the enabling technique of VFs. Section 5 shows the exploitation of the VF concept to support V&V activities. Section 6 presents benefits and limitations, while Section 7 summarizes the paper and defines future work.

2 RUNNING EXAMPLE

This running example builds upon previous work [24], where we focused on a small subset of the complete safety-critical adaptive cruise control (ACC). The ACC is an advanced driver assistance system (ADAS) which is already widely available in commercial road vehicles but is interesting as a system-under-study because it is one of the precursors of fully autonomous vehicles. ACC is a (radar-based) system, which is designed to enhance driving comfort and convenience by relieving the driver of the need to continually adjust his speed to match that of a lead vehicle.

Figure 2 presents the two modes of the ACC. In the *speed control* mode, the goal is to control the ego vehicle such that the velocity of the ego vehicle is maintained as the desired

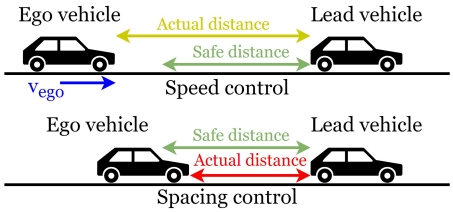


Figure 2: Concept view of the adaptive cruise control [15]

velocity set by the driver ($V_{ego} = V_{set}$). The second *spacing control* mode occurs when the ego vehicle is within the safe distance to the lead vehicle. Here, the goal is to control the ego vehicle such that the relative distance between the ego and lead vehicle is held constant to the user-defined safe distance ($D_{dist} = D_{safe}$). The ACC algorithm automatically switches between modes. This way, the spacing control is activated by approaching a lead vehicle and in case the lead vehicle disappears, e.g. by sudden acceleration or lane change, the ACC switches back to the speed control, which will control the ego vehicle to again adhere to the user-defined desired velocity.

The ACC is clearly safety-critical, as a failure of this component or sub-components could lead to an unsafe control behavior, possibly causing material damage and/or physical injuries if the system fails to brake safely. The key function of the ACC is maintaining a safe distance to the lead vehicles. Within our setup, the speed of the vehicle is controlled to ensure this proper vehicle-to-vehicle distance. This is implemented within an advanced control algorithm and monitored by a safety function (SF) to ensure this safe distance.

The *non-safety critical* and *safety critical* parts of the application are decoupled as much as possible within the application architecture. This enables a high degree of independence, both in the development and the V&V process stage of both parts of the application. The non-safe part is a model predictive control (MPC) algorithm enabling the ACC functionality, inspired by a Mathworks® example[1]. The MPC is a form of control in which the current control action is obtained by solving, at each sampling instant, a finite horizon open-loop optimal control problem using a physics-based model [12]. The conceptual architecture of the model-predictive ACC controller is shown in Figure 3.

Within this control algorithm the physical behavior of the car is modeled within the model-of-the-physics, which is used within the optimizer to optimize the acceleration/deceleration of the ego vehicle. In this use case, the correct behavior of the system, namely controlling the spacial distance between the ego vehicle and the lead vehicle, heavily depends on the correct predictions of the physical behavior of the vehicle under control. There exists two challenges: (i) the model-of-the-physics needs to match the interface of the control algorithm, meaning that the model interface needs to match with the optimizer algorithm and (ii) the modeled physical

[1] https://www.mathworks.com/help/mpc/ug/adaptive-cruise-control-using-model-predictive-controller.html

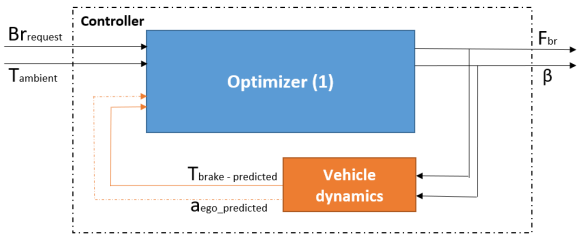


Figure 3: Conceptual architecture of MPC

behavior within the model-of-the-physics need to match as close as possible with the real behavior of the car under control. It is therefore key to reason about (i) the model interface for correct integration and (ii) the model fidelity of the model-of-the-physics to ensure "correct" (and safe) behavior predictions.

The safe part of the application is the functional safety mechanisms, ensuring safe behavior of the ACC. The following safety function is in scope of this work, as it include both traditional safety mechanisms using **fixed safety bounds**, and safety mechanisms using models-of-the-physics possibly using **dynamic safety bounds**:

- **SF3 - Safe Distance:** This safety mechanism ensures that the safe distance, either fixed or dynamically calculated, is never violated by the ego car. This ensures the safety of both the lead and ego vehicle.

Under normal conditions, the optimizer predicts the braking distance of the ego car using a model-of-the-physics of the physical behavior of the vehicle and the thermodynamics of the brakes. This allows the safe distance safety bounds to be dynamically adapted depending on the available brake momentum of the ego vehicle. Under faulty conditions however, a fall-back mechanism is activated and a fixed safe distance is determined. Therefore, accurate predictions of the physical behavior of the system is crucial to ensure safe ACC functionality. This includes explicit reasoning about how well some properties of the system are predicted by the model-of-the-physics, such as disc brake temperature or brake slip. As discussed in Section 4.2.3, validity frames (VF) contain the information and processes required to ensure that models are used within a context wherein they reflect reality, as well as a measure of this fidelity.

3 LITERATURE REVIEW

There exists some confusion in the related literature about the meaning of the terms "verification" and "validation". Within this work, we stick to the traditional definition [2, 8], such that *verification* aims to prove that the implementation fulfills its specifications and *validation* aims at increasing confidence in the correct operation of an implementation. As the complexity of CPSs has increased, so has the complexity of the corresponding V&V processes, which has been addressed in the literature in various ways. This ranges from research

on the overall V&V processes [22] to investigating new V&V methods and techniques [19] to optimize individual activities.

In earlier work [24], we facilitated the model-based V&V process of safety-critical systems by enriching the knowledge about system components with V&V information. Specifically, components implement *artifacts* which are inputs to V&V *activities*. *Process rules* are then used to (semi-)automatically generate V&V processes. In this work, we utilize the same technique on the information captured in validity frames, as discussed in Section 4. In addition, this work focuses on obtaining proof of correct and deterministic component behavior under both normal and faulty conditions, which is as important as or even more important than the component implementation itself.

The frames concept is not a novelty but has been around since the early 1980's when Zeigler [26] defined the original "experimental frames" idea. These experimental frames (EF) helped documenting the meta-information necessary to execute the model itself. Traore and Mazy [21] formalized this EF concept and Schmidt *et al.* [20] showed the use of EFs within the model-based testing of simulation models. Denil *et al.* [5] observed that a model's frame depends on the activity that is performed and describes why different activities require different frames. They proposed the VF concept which defines the experimental context of a model in which that model gives predictable results.

In previous work [23], we made this VF concept tangible and showed a basic use case within the development of CPSs. In particular, we use VFs to capture the range of validity for models-of-the-physics for a CPS, which assists with model reuse and creating reduced-order models with sufficient fidelity. The current work extends the use of the VF concept by introducing it in the V&V process for both reasoning about the overall V&V processes and optimizing individual V&V activity methods. Authors in [6, 13] used knowledge about the system under test (SUT) to prune the search space and reduce testing effort.

4 VALIDITY FRAMES CONCEPT

This section will introduce the VF concept, its objective, scope, and structure with focus on V&V activities.

4.1 Validity Frame Objective and Scope

Extending the work of [5], we formalized the VF concept [23] to define the range-of-validity of a component. More specifically, a VF is used to (i) capture the range-of-validity of a model and (ii) provide methods/processes to assure that a model faithfully represents the source system. This way, the VF is not only be used to *reason about the range-of-validity of a model*, but can also be used to reason about the use of a model in a new or altered context, not captured within the VF. The latter is enabled by the enclosed processes of the VF, enabling model validation. This increases the model re-useability by enabling exploration of model usage within different known or unknown contexts.

A concrete example of the usage of VFs is found in our earlier work [25]. In that work, the range-of-validity for models of electrical resistors is captured, to define the temperature and voltage ranges where Ohm's law sufficiently represents the real-world behaviour of a resistor. This information is then used in design-space exploration (DSE), where it can be detected that the simple Ohm's law resistor model is used outside of its validity range and should be replaced by a more detailed resistor model.

The focus of the VF concept for the present work is on reasoning and performing (i) correct simulation experiments with the comprising models and (ii) correct deployment of the comprising models. That is, ensuring that a model is used within the context in which it (sufficiently) matches reality. If a model is simulated within or deployed to a context where it is not valid, then it is useless (or dangerously inaccurate and unsafe). We therefore capture the range-of-validity of a model in a corresponding *VF package* in which all relevant validity knowledge of that model is packed together in a structured way. This implementation detail is beneficial in the domain of deployment for CPSs, but may be altered to generalize the VF concept to other domains.

4.2 Validity Frame Structure

A VF is conceptually divided in three main parts[2]:

- **Meta-data:** Meta-information usable as input for DSE algorithms
  - Model structure
  - Execution environment configuration
  - Model fidelity
  - V&V evidence
- **Operational:** Model scope and usage context
  - Equivalence classes
- **Process:** Processes to perform experiments, calibrate models and assess the model validity
  - Main experiment building blocks

Each of the above mentioned information will be discussed in detail in next subsections and the use will be demonstrated in Section 5.2.

4.2.1 Meta-data. The subsequent subsections describe the essential meta-data captured within the VF of a component, which are used within the V&V stage of the design process.

Model Structure. The first part of the VF concept is capturing the meta-information on the **structure** of the models, meaning the components and their (inter-) relationship. These structure variables are divided into three groups:

- **Ports:** Entities representing the *interface definition* of the model
- **State variables:** Entities representing the partial state of the model
- **Calibration parameters:** Entities representing the calibratable elements of the model

[2]We omit here a meta-model representation of the validity frame concept for space reasons. It can be found in our earlier work [23].

For each of these structure variables, the most essential information is captured e.g. *parameter-based valid boundaries*, *datatype*, *unit*, etc. Range information can either be provided by the user based on his/her experience or can be extracted from experiment results. The user can also add value's distribution to each port. Below, the partial description of the *DistanceSignal* output port is shown. It shows that the output signal on this port is guaranteed to be in range of 0 to 100 m, with *double* as the datatype.

```
<Port Name="DistanceSignal" RangeMax="100.0" RangeMin="
    0.0" Type="Out" Unit="m" Datatype="double" />
```

Execution Environment Configuration. The next part of the VF concept is the meta-information captured for the **execution environment**, which is the environment in which the model is used. Within the current scope of the VF concept, a VF can contain both models with the *intent of simulation* and the *intent of embedded deployment*, e.g. use within embedded controller. We therefore explicitly capture meta-data on the execution environment depending on the purpose of the model. Depending on the execution environment, different aspects will have influence on the validity of the model. More specifically, reasoning about the (numerical) accuracy of the model requires the capturing of essential meta-information of the **simulation environment**:

- **Solver type:** Continuous- or discrete-time solver
- **Solver method:** Identification of the solver method e.g. Dormand-Prince, Runge-Kutta
- **Step-size:** The sampling time at which a model is updated. This step-size, either *fixed* or *adaptive*, will have great influence on the correctness of the simulation and the stability properties

As well, essential meta-information of the **embedded environment** is captured:

- **Processor clock:** The maximum clock frequency
- **Memory:** The maximum available memory

For the embedded execution environment, meta-data on the embedded platform is captured as well as on the methods and techniques used to transform the simulation model into a deployable model e.g. compiler and optimization flags. Below, the partial description of the compiler used to compile the model for the NXP MPC 5744p safety-critical embedded platform is shown. This can be used for HiL setup and simulation.

```
<Compiler Name="Freescale C/C++ Compiler" Executable="
    gcc" Options="-C99 --stdl --no_exceptions" />
```

Model Fidelity. The next crucial part of the VF concept is **model fidelity** meta-data, which provides a metric of how well a model emulates its source system counterpart. A **model validation process** needs to be performed as in Section 4.2.3, resulting in one or more metric results specifying model fidelity. Note this fidelity metric heavily depends on the execution environment of the models, either simulation or embedded, as (i) a continuous time system cannot be simulated on a digital computer without proper numerical integration methods, which approximates the continuous behavior [26], and (ii) a continuous time system cannot be executed on a clocked processor without discretization of the model's continuous behavior. Therefore we need to capture not only the fidelity metric in isolation, but also its dependencies with the model structure and execution environment. Below, the partial description of the model fidelity of the model-of-the-physics used in the MPC algorithm is shown. It shows that we use two metrics to specify the model fidelity. The *Root Mean Square error* and the *Cross-correlation* is the result of comparing the results of a real-world experiment using the source system against the results of a representative virtual experiment on the model under investigation.

```
<AccuracyMetrics>
  <Metric Name="RMSE" Value="0.55" Port="Velocity" />
  <Metric Name="XCOR" Value="39" Port="Velocity"/>
</AccuracyMetrics>
```

Verification and Validation. The next meta-information stored in the VF is the evidence of the conducted/performed V&V activities. Earlier work [24] explicitly captured knowledge about the level of V&V activity completion for each component within the *artifact-view* of a component, as a precursor concept of the VF concept where we now explicitly capture V&V activity information.

4.2.2 Operational. Within the V&V part of the VF concept, we also allow defining **equivalence classes** (EQ). Engineers can partition their input boundaries to identify different classes within the input boundaries that give a similar/common output behavior of the model or partition based on different modes of the system. Below, the partial description of the EQs of the safety mechanism SF3 (from Section 2) are shown. The behavior of SF3 can be divided into two main groups (based on the two ACC modes): EQ1 where the distance to the lead vehicle is small (0-25 meters), and EQ2 where the distance is large (26-100 meters).

```
<EquivalenceClassCfgDescription>
 <EQDescription Name="Q1" RangeMin="0" RangeMax="25"/>
 <EQDescription Name="Q2" RangeMin="26" RangeMax="100"/>
</EquivalenceClassCfgDescription>
```

4.2.3 Processes. Another key part of the VF concept is the contained methods/processes which enable (i) the correct use of the model within its correct context, (ii) assessment of the model fidelity, and (iii) custom-fit the model for a specific intent/purpose via model calibration. These processes are crucial within the V&V stage where the implemented behavior of the system under design needs to be verified and validated against the expected behavior. Within the VF concept, two main processes are included:

- **Back-to-back (B2B) process:** The **validation process** to assess how well a model emulates its source system, taking the model purpose into account. Essentially, the

purpose of the model is translated into a real-world experiment, conduced on the source system and to a representative virtual experiment, performed on the model under study. The resulting information of both experiments are used as input for the function(s) to determine the model fidelity.
- **Calibration process:** The process of custom-fitting models by altering the calibration parameters. Essentially, after calibrating the model, the B2B process is used to validate the calibrated model.

The main building blocks of these two processes can be re-used to conduct experimental V&V activities:

- **Test-case design:** The definition of the experiment's test-cases or experimental traces, based on the experiment objectives, provided as input to the (virtual) system. The *model structure*, *equivalence classes* and *previously conducted virtual experiments* are the key inputs to semi-automatically generate valid test-cases.
- **Execution environment setup:** The definition of the experimental harness used to perform correct experiments on the (virtual) system together with the specification of the execution environment capable of executing the (virtual) system to generate its behavior. For the experimental harness, an integration model is generated which connects the stimuli and assessment elements to the component-under-experiment. The key inputs that enable this are the *model structure* and the *execution environment configuration*.
- **Experiment execution:** The process of conducting the defined experiment or experiment suite. This gives the experimental results used for further assessment.
- **Experiment assessment:** The assessment process for observing and analyzing the virtual system output to observe if the experimental conditions are met. Key inputs to generate these assessment functions are the *model structure*, *system requirement* and the *experiment objectives*. Note that typically a tolerance bound is defined on the expected behavior. This tolerance bound can be explicitly defined within the VF or can be derived from system specifications.

By using the meta-data available within the VFs of models, almost the complete experiment process from *test-case design* to *experiment assessement* at unit level can be (semi-) automated. Besides the use at unit level, the VF data can also ease the V&V activities at integration level, e.g by providing automated interface checks[3] of interconnected components. This can largely reduce the effort and cost to conduct experimental V&V activities. This is demonstrated in Section 5 by conducting different V&V methods on the running example of Section 2. Note that the *Experimental Frame* (EF) of Zeigler [26] is also defined as the conditions under which a system is observed or experimented with. The test-case design process resembles the *generator* of Zeigler, while the experiment assessment resembles the merge of the *transducer* and *acceptor* parts.

[3]Checking the interface compatibility in terms of matching units, datatypes and boundary ranges

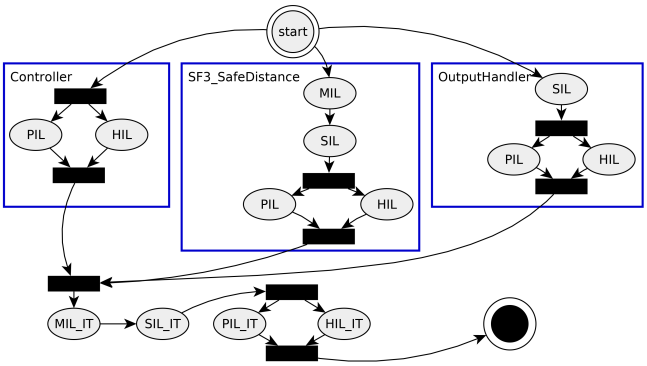


Figure 4: Auto-generated model-based V&V test process

5 VALIDITY FRAMES SUPPORTING VERIFICATION AND VALIDATION

This work aims to ease the V&V activities involved within a model-based design process by using the data explicitly captured within the VF concept of the comprising system components. Parts of the running example of Section 2 will be used in this section to demonstrate this efficiency gain. The following assumptions are made:

- **Assumption 1:** The system components were all developed and tested until a particular test level, depending on the use in previous use cases.
- **Assumption 2:** For each of the system components, a VF is defined comprising all essential information as discussed in Section 4.

5.1 Generation of Verification and Validation Processes

The first use of the captured V&V data for the different system components is using the *V&V activity completion knowledge*. In [24], we demonstrated that using this data, the model-based V&V process of the complete system is facilitated by (semi-) automatically generating valid and optimized V&V strategy processes. To enable the (semi-) automatic generation of valid V&V processes, we need a *model of the software architecture*, which defines the components and their interconnections, together with a set of rules or constraints that constrain the valid control flow of the model-based V&V activities.

The following rules are defined for the running example:

- V&V activities on the *software* level need to be performed sequentially after the V&V activities on the *model* level.
- All V&V activities on the *integration* level need to be performed sequentially after all V&V activities on the *unit* level of the comprising system components.

To generate the shown V&V strategy process in Figure 4, all components of the explicit modeled software architecture (namely *Controller*, *SF3_SafeDistance* and the *OutputHandler*) and their corresponding level of V&V activity completion are interpreted. Using the X-in-the-Loop (XiL) test strategy (MiL, SiL, PiL, HiL), the V&V activities that still need to be performed can be determined for each of the components and the overall integration (XiL_IT). The resulting set of V&V activities are used as input for the generation of a valid V&V strategy process. Using the process rules, not only the execution order of the V&V activities are specified, they also specify if a V&V activity can be excluded from the V&V strategy process, resulting in an optimized V&V process. This generated V&V strategy process can reduce test effort by (i) optimizing the V&V strategy to both the software architecture and any engineer-specific strategy and (ii) narrowing the experiment-space by eliminating unnecessary or duplicate V&V activities. The latter is particularly effort-cutting if the application architecture comprises re-usable (safety) system components, which are already verified and validated at a particular integration level.

5.2 Verification and Validation Software Testing

Once the VF meta-data for a complex system component is defined, this enhanced knowledge can be used to ease subsequent parts of the design process, such as the verification step. As mentioned in Section 1, the verification activities need to occur at different levels of abstraction, and all subsequently discussed V&V activities can be carried out at different stages of the model life cycle. Key differences are (i) the model under test itself and (ii) the execution environment setup. The preliminary function evaluation of the ACC controller is done at the MiL stage, with a complete virtual experiment environment running on a non-real-time host. The generated source code from the ACC controller is verified on the SiL stage, again using the same virtual experiment environment. Within the PiL stage, the generated object code of the controller is verified, running in the dedicated processor, interfacing with a virtual experiment environment running on a non-real-time host. Within the HiL stage, the embedded controller and the physical connections are verified by interfacing with a real-time target, enabling real-time performance of the virtual elements together with comprehensive, precise, and fast I/O capabilities. The VF concept comes into play in different parts in this XiL strategy: (i) the experimental traces and validity assessment function implementation, captured in the VFs, can be re-used within the different stages, (ii) the range-of-validity of the different representations of the *controller*, *model*, *source code*, and *object code* can be used for automating test-case design, (iii) the captured meta-data on the execution environment configuration can be used to automate the execution environment setup, and (iv) the captured meta-data on the specification of the transformations between the model representations can be used to automatically perform this transformation. That is, the object code can be generated from the source code using the compiler and linker specifications. Without the use of the VFs concept, this knowledge about the different representations, their range-of-validity, the execution environment setup and the transformation specifications would all be scattered implicit knowledge and manual actions would be required to use this info within the V&V activities.

5.2.1 Requirements-based testing. *Requirements-based testing* is the process of testing if a component fulfills the *functional* and *non-functional* requirements as defined in an official requirements document. The key value of VFs here is the re-use of conducted experiments and the automated experiment setup throughout the XiL process. The VF concept allows the capturing of each conducted experiment, both the experimental traces and the assessment function, within the process part of the component's VF. This automated re-use of requirements-based experiments can reduce the test-effort tremendously, as will be explored in future work.

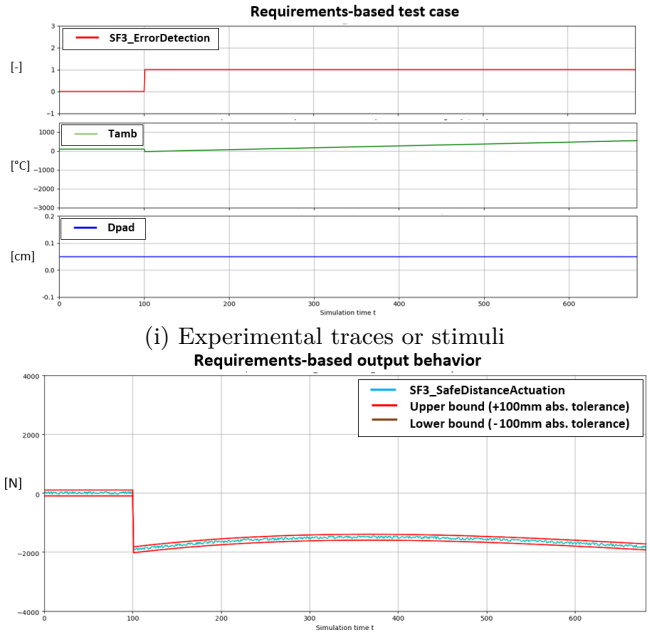


Figure 5: Evidence requirement-based test SF3

Figure 5 (i) shows a requirements-based experiment, (ii) shows the output traces, together with the identification of the expected behavior. This expected behavior can be explicitly defined within the assessment functions or can be retrieved by conducting the same experiment on higher fidelity models. As such, the retrieved expected behavior, together with some tolerance bound, can be used to assess whether or not the requirements-based experiment is passed.

The experimental setup links the experiment traces or stimuli to the SUT and verifies the SUT's behavior, and it is automatically generated along with the execution environment configuration. Figure 6 shows the experimental setup of

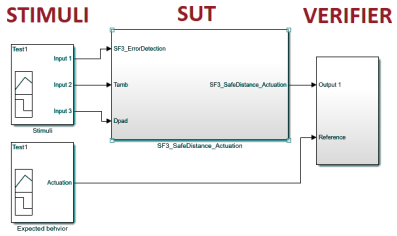
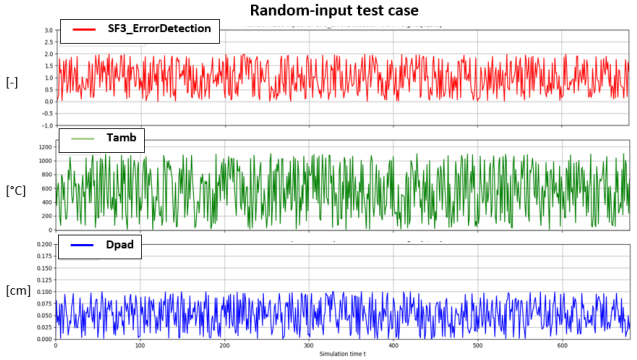


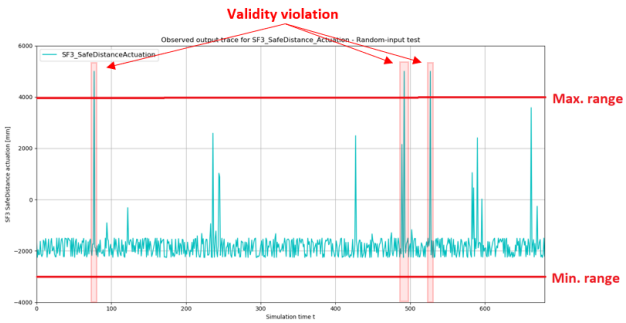
Figure 6: Auto-generated and configured experimental setup

the requirements-based experiment. As this is a MiL experiment, the configuration of the execution environment can be achieved by configuring the solver using the corresponding VF meta-data.

5.2.2 Random-input testing. *Random-input testing* methodology is the process of testing a component by selecting at random some subset of all possible input values. It is one of the least effective methodologies but can reveal some unexpected component errors [18]. The value of VFs is the use of the meta-data of the *model structure* to automate the test-case generation, where the *assumed ranges* and *equivalent classes* on the model inputs are used as input for the test-case generation. Again, the experimental setup and the execution environment configuration is automated.



(i) Experimental traces or stimuli

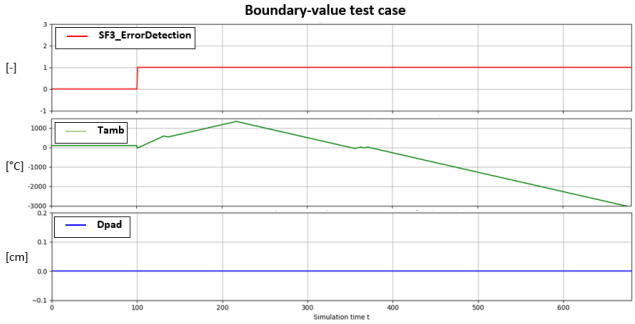


(ii) Observed output behavior

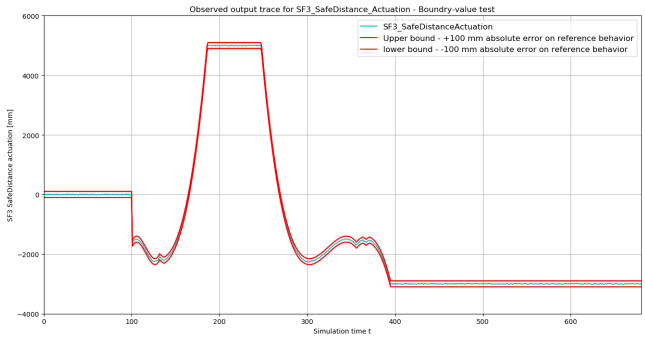
Figure 7: Evidence random-input test SF3

Figure 7 (i) shows the generated random signal values for each of the component inputs, (ii) shows the output traces, together with the identification of the expected behavior. This expected behavior is the *guaranteed range* for each of the component outputs which are captured in the model structure of the VF. Figure 7 (ii) clearly shows that the observed behavior of the SUT is not valid at 80, 490 and 520 ms, indicated by the red boxes, as the guaranteed output boundaries are violated. A *validity violation* can also be used as *termination condition*, meaning that a violation can terminate the experiment suite before completion.

5.2.3 Boundary-value testing. *Boundary-value testing* (or *interface testing*) is the process of testing the boundaries of the components behavior by selecting test points on and near the input boundaries. This method proves to be very effective as it is frequently observed that domain boundaries are particularly fault-prone [9]. The added value of VFs is again the use of the *assumed ranges* on the model inputs, captured within the model structure of the VF, to automate the test-case generation, the experimental setup and execution environment configuration.



(i) Experimental traces or stimuli



(ii) Observed output behavior

Figure 8: Evidence boundary-value test SF3

Figure 8 (i) shows the generated test points for the boundary-value test of input 1, (ii) shows the output traces, together with the identification of the expected behavior. This expected behavior can, as discussed before, be an explicit experiment trace or the result of the B2B process.

5.2.4 Fault-injection testing. The last V&V method in scope of this work is *fault-injection testing*. Fault-injection (FI) testing is the process of deliberately introducing faults in a component to analyze its behavior and response in faulty

conditions [4]. Fault-injection testing is typically performed within the V&V process of safety-critical applications where it is critical that system engineers not only prove that the component fulfills its functions, but that it can also gracefully handle hidden residual faults. The key value of VFs is the use of (i) *equivalent classes* and (ii) the meta-data of the *model structure* to automate the test-case generation and prune fault-space. More specifically, the VF data is not only used to automate the experiment setup and configuration of the execution environment, but also to automatically instrument the experimental setup with faults. This is performed by adding a *custom fault-injection annotation block* as seen in Figure 9. This annotation block was developed in previous work on model-implemented hybrid fault-injection [16], and is added at each position in the SUT where a fault needs to be injected. Depending on the VF data, this custom FI annotation block is configured to trigger corresponding fault patterns such as *stuck-to-zero* faults.

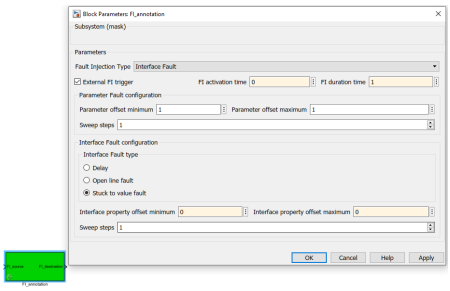
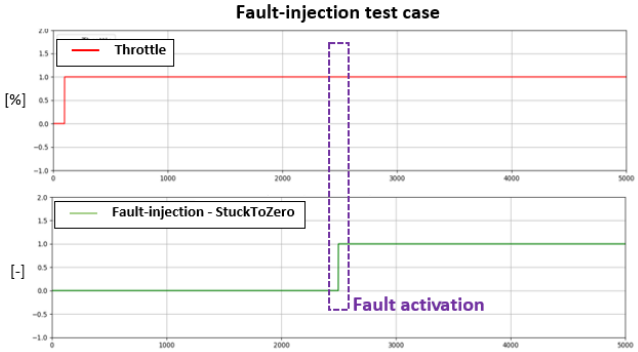


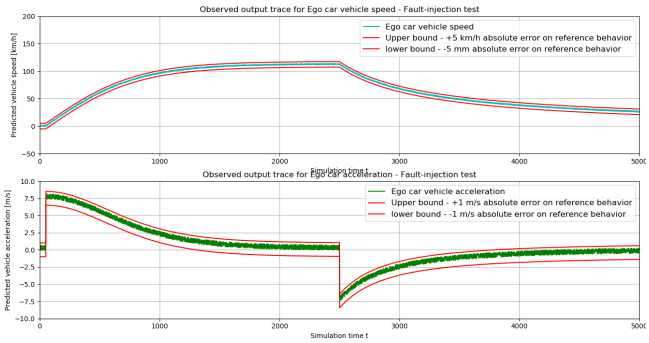
Figure 9: Custom fault-injection annotation block

As this model structure contains not only meta-data on the in- and outputs of the model but also on the parameters and/or state variables, this enables the injection of fault signals not only at the inputs of the model, but also in the model internals such as parameters and state variables. Also, *boundary ranges* of inputs and *equivalent classes* are used to set proper fault parameters and test cases [14, 15]. In this way, more test cases can automatically be generated, which results in a more efficient test strategy, increasing the confidence and reliability in the correctness of the component.

Figure 10 (i) shows the generated test points for the fault-injection test, while (ii) shows the output traces, together with the identification of the expected behavior. This expected behavior can, as discussed before, be an explicit experiment trace or the result of the B2B process. The expected behavior of the component-under-test shows that the component implementation contains a fault-handling mechanism, observed by the changing output trace at the FI time. Note that for each FI experiment, the *fault representativeness* [1] of the simulation-based fault experiment needs to be assessed. That is, checking the plausibility that a physical fault, e.g. short to ground, is represented in a correct way via the virtual experiment.



(i) Experimental traces or stimuli



(ii) Observed output behavior

Figure 10: Evidence fault-injection test car dynamics

6 DISCUSSION

Benefits. By using the enhanced model knowledge captured within the corresponding VFs, the V&V process can be made more efficient by customized and optimized the process and automating it partially. This decreases the V&V process effort drastically. By again capturing the V&V knowledge, gathered during an ongoing V&V process, and extending the corresponding VFs, the (re-)usability of these models can further be extended. This can potentially decrease the V&V process effort of new projects even further e.g. by re-use of defined test scenarios or automatically skipping tests at a particular test phase.

Limitations. There are some limitations associated with our approach. The first limitation is that the effort reduction within the V&V part of the design process is the result of an increased effort within the design/development part of this process, so the net benefits of using the proposed VF concept in V&V processes will become most apparent as the re-use of components increases. Second, the test automation using the VF data assumes that the XiL validation tool chain is well-known and fixed. Lastly, the quality and usability of the conducted tests heavily depend on the correctness and completeness of the captured VF data. Faulty or incomplete VF data will result in nonsensical or failed tests.

7 CONCLUSIONS AND FUTURE WORK

In this work, we presented the use of the Validity Frame (VF) concept to support the V&V activities for complex cyber-physical systems. We first identified what meta-data is essential for facilitating (parts of) the V&V process and showed how this is captured within a structured way in the VF. Using an academic safety-critical cyber-physical system, which comprises multiple models-of-the-physics, we demonstrated the use of this meta-data to (i) automate the generation of a valid V&V process, (ii) automate the setup and execution of some of the V&V activities, (iii) front-load V&V activities at earlier design phase, and finally (iiii) lower the time and cost of the testing process.

Within this work, we have ignored the existing relation between the conducted V&V experiments and the (set of) *properties-of-interest* that is/are validated or verified using the experiments. In the future, we plan to make this relation between the experiments and the properties-of-interest explicit and capture it within the VF. We also plan to formally specify the VF concept and broaden their use within state-of-the-art engineering processes, focusing on model-based design.

ACKNOWLEDGMENTS

This work has been carried out within the framework of ITEA 3, EUREKA Cluster programme project 'EMPHYSIS' (Embedded systems with physical models in the production code software, HBC.2017.0290) funded by the agency Flanders Innovation & Entrepreneurship (VLAIO).

REFERENCES

[1] Jean Arlat, Yves Crouzet, Johan Karlsson, Peter Folkesson, Emmerich Fuchs, and Günther H Leber. 2003. Comparison of physical and software-implemented fault injection techniques. *IEEE Trans. Comput.* 52, 9 (2003), 1115–1133.

[2] Barry W Boehm. 1984. Software engineering economics. *IEEE transactions on Software Engineering* 1 (1984), 4–21.

[3] Christopher Brooks, Chih-Hong Cheng, Thomas Huining Feng, Edward A Lee, and Reinhard Von Hanxleden. 2008. Model engineering using multimodeling. In *Proceedings of the 1st International Workshop on Model Co-Evolution and Consistency Management (MCCM'08)*.

[4] Domenico Cotroneo and Roberto Natella. 2013. Fault injection for software certification. *IEEE Security & Privacy* 11, 4 (2013), 38–45.

[5] Joachim Denil, Stefan Klikovits, Pieter J Mosterman, Antonio Vallecillo, and Hans Vangheluwe. 2017. The experiment model and validity frame in M&S. In *Proceedings of the Symposium on Theory of Modeling & Simulation*. 1–12.

[6] Christian Dietrich, Achim Schmider, Oskar Pusz, Guillermo Payá Vayá, and Daniel Lohmann. 2018. Cross-Layer Fault-Space Pruning for Hardware-Assisted Fault Injection. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1109/DAC.2018.8465787

[7] Sanford Friedenthal, Regina Griego, and Mark Sampson. 2007. INCOSE model based systems engineering (MBSE) initiative. In *INCOSE 2007 symposium*, Vol. 11.

[8] Lex Heerink and Ed Brinksma. 1995. Validation in context. In *International Conference on Protocol Specification, Testing and Verification*. Springer, 221–236.

[9] Bingchiang Jeng and Elaine J Weyuker. 1994. A simplified domain-testing strategy. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 3, 3 (1994), 254–270.

[10] Edward A. Lee. 2008. Cyber Physical Systems: Design Challenges. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC) (00, Vol. 3)*, Reginald N. Smythe and Alexander Noble (Eds.). IEEE, 363–369. https://doi.org/10.1109/ISORC.2008.25

[11] Edward A Lee. 2017. *Plato and the Nerd: The Creative Partnership of Humans and Technology*. MIT Press.

[12] David Q Mayne, James B Rawlings, Christopher V Rao, and Pierre OM Scokaert. 2000. Constrained model predictive control: Stability and optimality. *Automatica* 36, 6 (2000), 789–814.

[13] Xiankai Meng, Qingping Tan, Zeming Shao, Nan Zhang, Jianjun Xu, and Haoyu Zhang. 2018. Optimization methods for the fault injection tool SEInjector. In *2018 International Conference on Information and Computer Technologies (ICICT)*. 31–35. https://doi.org/10.1109/INFOCT.2018.8356836

[14] Mehrdad Moradi, Cláudio Gomes, Bentley James Oakes, and Joachim Denil. 2019. Optimizing Fault Injection in FMI Co-Simulation through Sensitivity Partitioning. In *Proceedings of the 2019 Summer Simulation Conference* (Berlin, Germany) *(SummerSim '19)*. Society for Computer Simulation International, San Diego, CA, USA, Article 32, 12 pages. https://doi.org/10.5555/3374138.3374170

[15] Mehrdad Moradi, Bentley James Oakes, Mustafa Saraoglu, Andrey Morozov, Klaus Janschek, and Joachim Denil. 2020. Exploring Fault Parameter Space Using Reinforcement Learning-based Fault Injection. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. 102–109. https://doi.org/10.1109/DSN-W50199.2020.00028

[16] Mehrdad Moradi, Bert Van Acker, Ken Vanherpen, and Joachim Denil. 2019. Model-Implemented Hybrid Fault Injection for Simulink (Tool Demonstrations). In *Cyber Physical Systems. Model-Based Design*, Roger Chamberlain, Walid Taha, and Martin Törngren (Eds.). Springer International Publishing, Cham, 71–90.

[17] Peter Mosterman. 2007. Model-Based Design of Embedded Systems. In *2007 IEEE International Conference on Microelectronic Systems Education (MSE'07)*. 3–3. https://doi.org/10.1109/MSE.2007.65

[18] Glenford J Myers, Corey Sandler, and Tom Badgett. 2011. *The art of software testing*. John Wiley & Sons.

[19] Rakesh Rana, Miroslaw Staron, Christian Berger, Jörgen Hansson, Martin Nilsson, and Fredrik Törner. 2013. Increasing Efficiency of ISO 26262 Verification and Validation by Combining Fault Injection and Mutation Testing with Model based Development. *ICSOFT* 2013 (2013), 251–257.

[20] Artur Schmidt, Umut Durak, and Thorsten Pawletta. 2016. Model-based testing methodology using system entity structures for MATLAB/Simulink models. *SIMULATION* 92, 8 (2016), 729–746. https://doi.org/10.1177/0037549716656791

[21] Mamadou K. Traore and Alexandre Muzy. 2006. Capturing the dual relationship between simulation models and their context. *Simulation Modelling Practice and Theory* Volume 14, Issue 2 (2006), Pages 126–142.

[22] FW Vaandrager. 2006. Does it Pay Off? Model-Based Verification and Validation of Embedded Systems! *PROGRESS White papers 2006* (2006), 43–66.

[23] Bert Van Acker, Paul De Meulenaere, Joachim Denil, Yuri Durodie, Alexander Van Bellinghen, and Kris Vanstechelman. 2019. Valid (Re-)Use of Models-of-the-Physics in Cyber-Physical Systems Using Validity Frames. In *2019 Spring Simulation Conference (SpringSim)*. IEEE, 1–12. https://doi.org/10.23919/SpringSim.2019.8732858

[24] Bert Van Acker, Joachim Denil, Paul De Meulenaere, Bjorn Aelvoet, Dries Mahieu, and Jan Van Den Oudenhoven. 2018. Generation of test strategies for Model-based Functional Safety testing using an Artifact-centric approach. In *Proceedings of MODELS 2018 Workshops Copenhagen, Denmark, October, 14, 2018/Hebig, Regina [edit.]*. 563–569.

[25] Simon Van Mierlo, Bentley James Oakes, Bert Van Acker, Raheleh Eslampanah, Joachim Denil, and Hans Vangheluwe. 2020. Exploring Validity Frames in Practice. http://msdl.cs.mcgill.ca/people/bentley/research/VanMierlo2020.pdf. *Proceedings of International Conference on Systems Modelling and Management (ICSMM* (2020). Accepted.

[26] Bernard P Zeigler, Tag Gon Kim, and Herbert Praehofer. 2000. *Theory of modeling and simulation*. Academic press.