Journal of Information Processing Vol.25 797–810 (Aug. 2017)

[DOI: 10.2197/ipsjjip.25.797]

Regular Paper

Safety Verification Utilizing Model-based Development for Safety Critical Cyber-Physical Systems

Tasuku Ishigooka[1,a)] Habib Saissi[2,b)] Thorsten Piper[2,c)]
Stefan Winter[2,d)] Neeraj Suri[2,e)]

Received: November 18, 2016, Accepted: May 16, 2017

Abstract: The application of cyber-physical systems (CPSs) in safety-critical application domain requires rigorous verification of their functional correctness and safety-relevant properties. We propose a practical verification process which enables to conduct safety verification of safety critical CPSs. The verification process consists of (a) a system model construction method, which generates a system model by combining software described in C and plant model code reused from model-based development, (b) a model transformation method, which transforms the plant models including differential algebraic equations (DAE) to approximate models without DAE to reduce verification complexity induced by DAE solver execution, (c) a model simplification framework, which enables the simplification of bond-graph plant models using domain-knowledge-based replacement of complex model components for further verification overhead reductions, and (d) a formal verification based on symbolic execution. We implemented the proposed methods and framework, and successfully applied the proposed verification process for safety verification of automotive brake control systems. The results of the study demonstrate that the verification detects a complex failure condition in a real-world brake control system from the generated system model and that the automated model transformations of the CPS models yield significant verification complexity reductions without impairing the ability to detect unsafe behavior.

Keywords: symbolic execution, model-based development, model transformation, bond-graph, signal-flow graph, automotive cyber-physical systems

1. Introduction

A cyber-physical system (CPS) is an embedded control system that strongly links computing and physical systems [1], [2]. For example, automotive safety-critical CPSs consist of controllers (cyber), called electronic control units (ECU), and control targets, such as mechanical components (physical), called plants. The ECU software measures plant behavior through sensors and controls actuators by issuing control commands in real time in accordance with the sensed state of the plant. Automotive safety-critical CPSs implement a highly collaborative control process between electronic and mechanical components.

Automotive CPSs have stringent safety requirements, because system failures may cause critical damage to users. Therefore, the development process for CPSs requires rigorous verification steps. In automotive CPSs, the model based development (MBD) approach prevails. The approach requires controller models, which execute discrete processing, and plant models, which have continuous behavior based on the physical laws, as shown in **Fig. 1**. Concretely, the controller model comprises control algorithms described by ordinary differential equations (ODE) and the plant model replicates physical control target behavior described by differential algebraic equations (DAE) for the energy conservation theorem, in which power of action to the physical object and power of reaction from the object are equal.

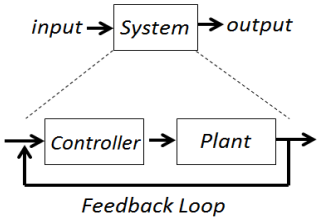


Fig. 1 Control system.

These models are used to validate the system design of automotive CPSs prior to building an actual prototype. The system behavior is simulated by numeric solvers, such as ODE solvers and DAE solvers. However, it is difficult for users to identify subtle design faults which occur only upon rare combinations of specific conditions, because these constitute a vanishingly small fraction of all possible test cases. This makes the selection of such cases in the testing process very unlikely. Nevertheless, if these rare conditions occur during operation of the system, any unidentified defects can severely threaten the safety of its users.

While the requirement for safety verification for critical CPSs is easy to state, i.e., ensuring coverage of all possible computing and control interactions, it is infeasible to achieve us-

[1] Hitachi Ltd., Hitachi, Ibaraki 319–1292, Japan
[2] Technical University of Darmstadt, Darmstadt 64289, Germany
[a)] tasuku.ishigoka.kc@hitachi.com
[b)] saissi@cs.tu-darmstadt.de
[c)] piper@deeds.informatik.tu-darmstadt.de
[d)] sw@cs.tu-darmstadt.de
[e)] suri@cs.tu-darmstadt.de

ing conventional engineering approaches. Complicated control/computing interactions often result in timing and sequence malfunctions that are particularly hard to uncover. Additionally, as a design engineer often only has detailed knowledge of either the control or computing domain, resolving such issues is further complicated.

Differing from conventional statistical, experimental or simulation based approaches, the formal methods community has developed rigorous techniques, such as model checking, that target automated and comprehensive coverage of a system's states to discover complex malfunctions, such as timing faults. Amongst the varied model checking technologies, there exists an approach for hybrid system model checking which can express both discrete and continuous aspects [3], [4], [5], [6], [7]. However, hybrid system modeling is often not easy for most engineers because they are only conversant with their own design aspects, such as software or mechanical design.

Recently researchers have started investigating formal verification methods, which construct practical hybrid system models by combining achievements from different domains to conduct hybrid system verification [8], [9], [10]. Especially, methods, that combine model checking of software and simulation of plant models, are being studied. However, as the verification of control systems with analog input values has to explore an enormous state space, the verification method needs huge efforts and long verification time. It is difficult to achieve the verification, taking into account effects caused by differences of analog values, such as sensor values of system input or of signal timing. Consequently, the development of a verification method, which can verify system behavior/misbehavior across the entire range of possible inputs at all possible times is challenging.

As a suitable approach for such challenging scenarios, symbolic execution based formal verification is being advocated [11]. Verification properties are described in the form of assertion instructions. Once an unsatisfied assertion, i.e., a malfunction, is encountered, the symbolic execution engine analyzes the conditions under which it may occur and generates a test case with concrete input values that exhibit the detected defect during execution.

This paper proposes a practical verification process for safety critical CPSs. The process can find system level malfunctions by checking safety relevant properties of system models, which simulate control system behaviors by combining control software and plant source code, on the basis of symbolic execution based formal verification.

We assume that the plant code is generated from plant models developed for a simulation purpose. This triggers verification complexity due to DAE solvers for numerical calculation and sophisticated plant models. In order to address this issue, our proposed verification process contains a plant model transformation method and a model simplification framework to reduce the complexity of CPS verification.

In summary, this paper makes the following contributions to the state of the art in CPS verification.

- Construction of a practical verification process for safety critical CPS.
- Design of a system model construction method for safety verification based on symbolic execution.
- Development of a plant model transformation procedure that eliminates the need of DAEs.
- Construction of a plant model simplification framework to support the plant model simplification, which reduces the computation load by domain-knowledge-based replacement of complex model components and approximation of the model behavior by model parameter configuration based on feedback of simulation results.
- Implementation and evaluation of these proposed methods and framework using two case study examples from the automotive industry.

This paper is based on Refs. [12], [13] and is organized as follows: In Section 2, we provide a background. Section 3 describes our proposed verification process and methods. Section 4 and Section 5 present experiment results through two case study examples. Section 6 discusses effects of the proposed methods. Section 7 introduces related work. Section 8 concludes this paper.

2. Background

We start by describing the target CPS's using a concrete example in Section 2.1, before we present a model-based development in Section 2.2. A general symbolic execution based formal verification is explained in Section 2.3.

2.1 Target Cyber Physical System

Figure 2 shows our target CPS. It consists of one or more controllers, one shared plant, either an operator or the environment, or both. The controller senses maneuver events of a human operator, such as braking operation, and events, which are generated according to changes in the environment, such as sideslip. We assume that these maneuver events and environment events may vary on seconds scale. This is a limitation of our approach. However, that is valid for many scenarios, for example events generated by a human operator.

To discuss the specifics of CPSs modeling, we consider the simplified automotive brake control system, which satisfies the assumption for our target CPSs, presented in **Fig. 3**. The brake control system produces a brake force in accordance with the moving amount of the brake pedal stroke operated by a driver. As shown in Fig. 3, the system consists of mechanical components, such as brake calipers, which is a speed reducer, hydraulic circuits and motors, and electronic components such as ECUs controlling the motors. In the system, an ECU controls the brake force by monitoring the movement of the brake pedal stroke operated by

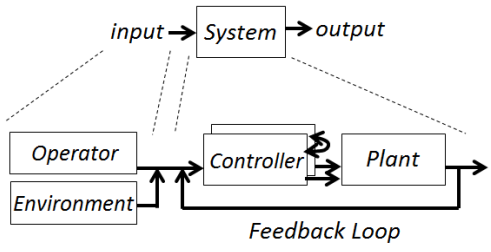


Fig. 2 Target CPS.

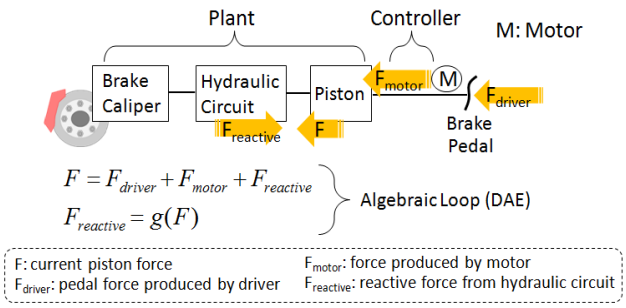


Fig. 3 An example automotive brake system.

the driver and assisting the pedal force by controlling the motor in synchronization with the moving amount of the stroke. The assistance enables users to lightly push the brake pedal.

The example of Fig. 3 shows how current piston force F is determined. Current piston force F is the sum of the pedal force produced by the driver F_{driver}, force produced by the motor F_{motor}, and the reactive force from hydraulic circuit $F_{reactive}$. $F_{reactive}$ is the result of applying function g with argument F. These formulae reflect the law of energy conservation resulting in the plant of the brake control system being modeled as equations with an algebraic loop. As a consequence, the plant model includes DAE, which impose a challenge to automated or computer-assisted verification.

2.2 Model-based Development

As we described in Section 1, a control system model consists of a controller model and a plant model. The controller model implements the system control logic and sends commands to the plant model which simulates the reactions of the physical component of the system. The combination of each model's sequential and iterative interactions constitutes the simulation of the system behavior.

Model-based development strongly supports the controller model design including discrete state transitions and the plant model design including ODE or DAE. The approach has been extensively used in the industry and been proven to be beneficial to the development of safety critical CPS.

There are modeling tools that aid the design of controller or plant models. For example, the controller model design is commonly conducted by signal-flow diagram based modeling tools such as MATLAB/Simulink® [14]. Simulink supplies code generation functions assuring automatic translation of the controller model into a runnable program. For the plant model design, bond-graph modeling tools such as AMESim™ [15], Simscape™ [16], and Modelica® [17] are predominant.

Bond-graph modeling enables users to design plant models by combining physical components such as spring, mass, and hydraulic circuit. Each component can be mapped to a real physical component and has a specific equation, such as a motion or constraint equation, and specific configurable parameters, such as weight, length, and so on.

As shown in **Fig. 4**, a bond-graph model consists of elements which translate to components, effort, flow and stroke. Effort and flow are domain-independent in bond-graph notation. For example, in the mechanical domain, effort means force and flow means

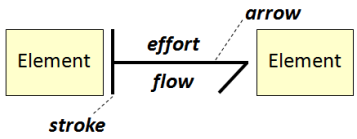


Fig. 4 Notation of bond-graph.

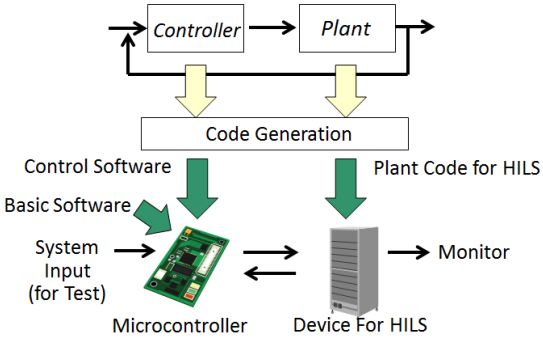


Fig. 5 Hardware-in-the-loop simulation process.

velocity. In the hydraulic domain, effort means pressure and flow means volume or flow rate. Stroke means flow direction. In the example, the flow value calculated by the left element is an input to the right one. The stroke reflects the causality in the calculation order of equations of each component. Finally, arrows describe energy direction.

Unfortunately, for a sophisticated behavior simulation of safety-critical CPSs solely ODE-based models are not sufficiently expressive. In many cases models of these systems contain DAE to reflect the energy conservation theorem. In numerical simulation, the plant behavior is simulated by leveraging a DAE solver. The DAE solver is executed at every calculation step in the simulation in order to find a set of suitable values of specific variables, such as a set of action and reaction forces by convergence calculation. Thereby, the DAE solver enables correct physical simulation but produces excessive computation load, which complicates automated verification beyond practicability.

The model-based development enables hardware-less system testing called hardware-in-the-loop simulation (HILS) as shown in **Fig. 5**. The HILS technology uses micro-controllers, which implement control software involving the controller logic and basic software, and special devices, which simulate the plant model, and some wiring to physically connect these components. The plant code is generated by discretization of the continuous plant model and translation into C source code. In the discretization process, the sampling rate is chosen according to the Nyquist sampling theorem [18] such that the equivalence between the continuous plant model and the resulting discretized model is guaranteed. HILS enables the engineers to test control systems with their actual product's control software using system inputs without real hardware and is extensively used in the industry. However, as typical HILS has no synchronization mechanism between the micro-controller and the special device for HILS, it is difficult to conduct a system test on exhaustive values, timings, and sequence of system inputs.

2.3 Symbolic Execution based Formal Verification

Symbolic execution based formal verification helps users to

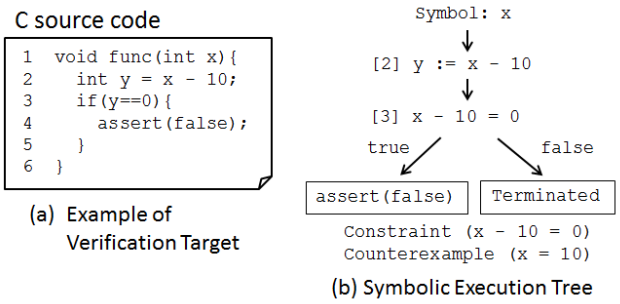


Fig. 6 Symbolic Execution based Formal Verification.

identify input values resulting in errors of target source code. The reason is that the verification can investigate all possible effects caused by changes in the values of variables, which users define as symbols. Consequently, if users define input variables of verification targets as symbols and insert assertions describing the verification property, the symbolic execution engine can find an input causing errors if the target does not satisfy the property specified in the assertions.

Figure 6 shows an example of symbolic execution based formal verification. In this example, we consider a function written in the C language (see Fig. 6(a)). The target conducts a simple calculation at line 2 and an assertion code which catches the error behavior by comparing the result of the calculation to the assumed error condition at line 3, in which case the assertion is violated (see line 4). As shown in Fig. 6(b), the symbolic execution based formal verification analyzes program logics of the verification target and extracts a formula describing constraints on the symbolic variables for specific paths in the code. In every step, the constraints are updated to describe the changes affecting the symbolic variables. When a branching statement is encountered, two constraint systems are created, one where the branching condition is evaluated to false and one where it's evaluated to true. Upon the execution of an assert statement the current constraint system is checked for satisfiability using a constraint solver. If it is not satisfiable, the search is resumed, otherwise an assignment of the symbolic variables is returned. The returned concrete values can be used as a test case to reproduce the found malfunction. Once all possible paths in the program are investigated, one can assume that the system satisfies the properties.

In HILS, DAE solver runs with upper bounds of the number of convergence calculation guaranteeing real-time performance. However, the use of the DAE solver increase the verification complexity because the symbolic execution tool has to analyze DAE solver code in addition to verification target. While approaches exist to remove DAE by formula manipulation of partial differentiation and substitution, they are difficult for engineers to manually apply without introducing errors into the transformed models and, thereby, threatening the validity of the verification.

3. Verification Process

We propose a formal verification process in Section 3.1. Our system model construction method of a verification target and property definition for safety verification are explained in Section 3.2 and Section 3.3, respectively. Then, we present a plant model transformation method and a plant model simplification framework in Section 3.4, and Section 3.5, respectively.

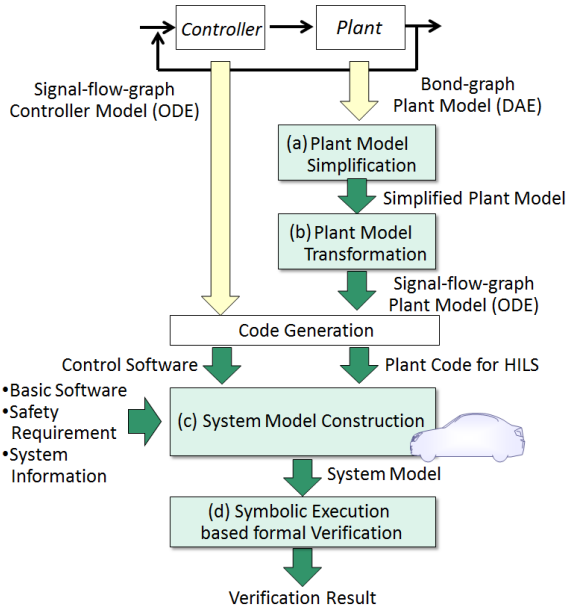


Fig. 7 Proposed verification process.

3.1 Overview

In order to reduce efforts for the safety verification, it is important to establish a verification process, which is compatible with the HILS process (Fig. 5) because the process enables the safety verification by combining design results from respective domains. We propose a verification process as shown in **Fig. 7**. Our proposed process enables a verification of the whole control system. In the process, a system model, which simulates target control system behaviors, is built by combining the control software, basic software, which is abstracted for microcontroller independent implementation, plant code, safety requirements, and system information, such as system inputs, task execution periods, and plant discretization time (see Fig. 7(c)). Next, the system model behavior is checked using symbolic execution based formal verification (see (d)). The system model construction is detailed in Section 3.2. The plant code is extracted from its HILS counterpart, excluding code which is dependent on the HILS emulation device.

In order to reduce verification complexity, the verification process also contains a plant model simplification phase (see (a)) and a plant model transformation phase (see (b)).

As Fig. 7 shows, a bond-graph plant model with DAE, which is an input file, is automatically simplified at the model simplification phase, which is presented in Section 3.5, and then is transformed into the signal flow plant model with ODE at the model transformation phase, which is presented in Section 3.4.

The purpose of safety verification is to prove CPS safety in specific situations where potential safety violations might occur. The specific situations means system state transitions, such as a state transition, in which a driver strongly pushes a brake pedal immediately after the brake pedal was released. We assume that the verification engineers verify individual safety properties. Although our verification approach employs bounded state search

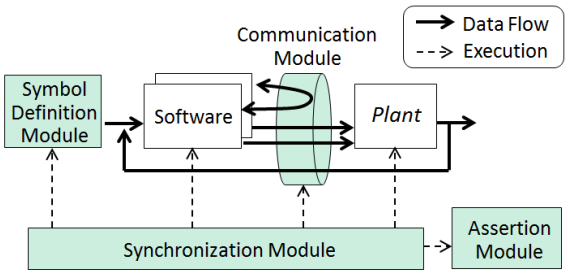


Fig. 8 System model structure.

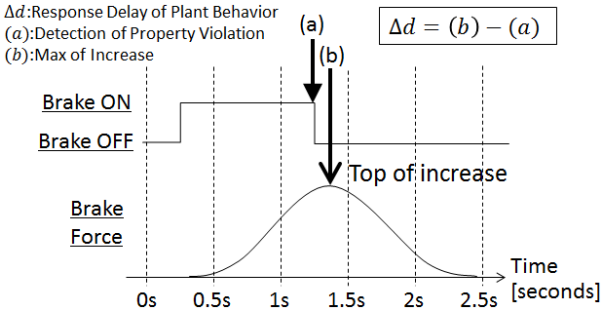


Fig. 9 Example of false positive caused by response delay of plant behavior.

algorithm, the verification is conducted in the verification time defined according to the time, which can cover the duration of the specific situation, such as the time to monitor system behavior change caused by system input change.

In our verification process, we conduct the model simplification before the model transformation because it is easier to simplify the bond-graph plant model than the signal-flow plant model.

3.2 System Model Construction

Figure 8 shows an overview on the system model structure. The integration of software and plant code into a common system model is not straightforward, as real-time constraints necessitate a synchronization mechanism for coordination of software and plant code. Therefore, the system model includes a communication module for data synchronization and a synchronization module for time synchronization.

Specifically, the communication module forwards the current control command, which the control software calculates for the actuator control, to the plant which receives the output value from the controller model.

Also, the module forwards the current sensor values, which reflect the behavior of the plant, to the control software which receives the output value from the plant model. The interaction between electronic control units (ECUs) is supported by the communication module as well.

The synchronization module maintains the current states of the software and the plant by sequential and iterative invocation at specific timings defined by task scheduling or update frequency of the plant state. Additionally, the module limits the verification time for feasibility. The reason is that an unlimited system model shows infinite behaviors because the software under control loops maintains continuous periodic invocations to control the plant. As presented in Section 3.1, the user has to determine the verification time. For example, if users want to see the system effect caused by the combination of exhaustive timings or sequence of system inputs, the bound is determined according to the time to check the change of the system behavior caused by the combination. Consequently, the bounded time should be defined taking into account properties of the target system.

For symbolic execution based formal verification, we developed symbol definition modules and assertion modules in the system model. The symbolic definition module defines system inputs such as user operations or events from the environment of the verification target as symbols. To monitor the exhaustive effect given by the system input value change, the module needs to redefine them. However, as the redefinition creates new symbols, the verification complexity increases. To avoid the frequent redefinition, we limit the redefinitions to specific timing, e.g., every 1 second or after the occurrence of a specific event. The redefinition frequency depends on the update frequency of system inputs of target CPSs. The effect is discussed using an example in Section 4.4. Additionally, as the system model updates its plant behavior at every discretization time, the decision of optimal discretization time is important for the verification complexity as well. The discretization time should be determined on the basis of the sampling theorem. The assertion module, which is an assertion code, checks properties of the target CPS by monitoring variables of the system model. The property definition method is detailed in Section 3.3.

3.3 Property Definition for CPS Verification

The property for CPS verification should carefully be defined taking into account response delays of actuator behaviors against system inputs or control commands from software because the plant behavior is affected by physical phenomenon. That means the property should include waiting time to check the property. Otherwise, the verification using the property will frequently return false-positives.

Figure 9 shows an example of a false positive in safety verification of an automotive brake control system. This example indicates that the verification using the property without waiting time causes a false positive detection. As the property of this example applies that unintended brake doesn't occur, the property is defined as a condition where the brake force doesn't increase when no braking is happening. However, as the brake force still increases even though the brakes are not pushed (see (a)) because of the response delay of the plant behavior (see (b)), the verification detects false positive. Consequently, it is required to include waiting time for the response delay (Δd) in the property definition to get rid of the false positives. In Section 4.3 we discuss our property definition method using an example.

3.4 Plant Model Transformation Method

Elimination of DAE from bond-graph plant models is known to be difficult because it entails the elimination of energy exchange in the bond-graph models which follows the law of conservation of energy. There is an approach in which the engineers design ODE models by manually transforming the DAE models, which requires deep knowledge about formulae translations in physics. We assume that test engineers take over the verifi-

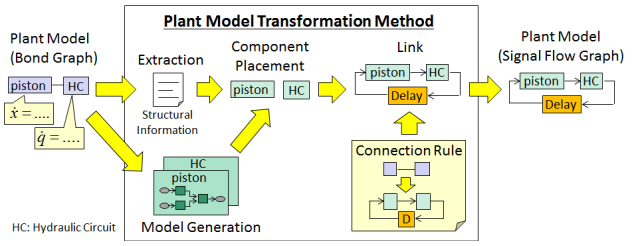


Fig. 10 An overview of automatic plant model transformation method.

cation process and conduct the safety verification as part of the system test process. Therefore, the engineers conducting safety verification cannot be expected to have the intimate knowledge required for manual model transformation. Thus, we propose a model transformation method, which transforms the bond-graph model to the signal-flow graph model and converts the model with DAE to ODE by adding delay blocks in between blocks with energy exchange.

A simple model transformation would replace a bond-graph model with an signal flow model. However, the transformed model will still contain the algebraic loop from bond-graph model. Our approach can eliminate the algebraic loop by inserting one-step-delay blocks into feedback loops between components. The purpose of delay blocks is to make the connected block use the signal value generated at the previous period. In the first calculation step, the delay block produces an initial value defined by the user. The initial value should be copied corresponding parameters of the components calculated at the beginning of the specific situation of simulation. Consequently, our proposed method can remove the algebraic loop from the plant model.

The delay block produces calculation errors at every simulation step. However, our evaluation results presented in Section 6.1 show that these errors remain negligibly small for the short simulation times commonly required for the assessment of individual safety properties.

In order to reduce the effort and avoid human errors during the model transformation, we propose an automated plant model transformation method which transforms bond-graph plant models to signal-flow plant models. **Figure 10** shows an overview of the automatic plant model transformation method. This method analyzes input files which store bond-graph plant models and extracts structural information on the plant models. In parallel, the method generates signal-flow subsystems according to the equation of each element in the bond-graph model. Each subsystem corresponds to exactly one element in the bond-graph model. The method places these subsystems according to the previously extracted structural information. Hence, it links each subsystem according to our proposed connection rule. Ultimately, this method outputs signal-flow graph plant models.

We defined connection rules of signal-flow-graph subsystems to preserve the energy flow direction with bond-graph component interactions. **Figure 11** shows our proposed connection rules. In bond-graphs, there are two data types, flow and effort. Moreover, there are two connection types, direct and multiple. The direct connection connects one element to another. The multiple connection involves more than 2 elements. The connection

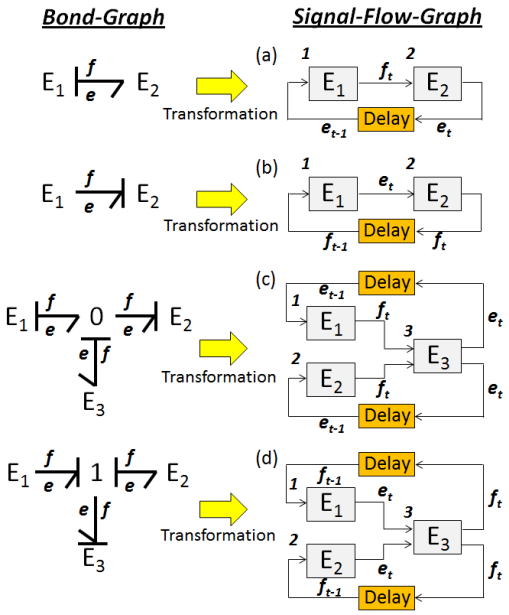


Fig. 11 Connection rule.

is implemented by a 0 junction or a 1 junction (see Fig. 11 (c) and (d)). This means that there are 4 connection relationships in bond-graph modelling [19]. Our proposed connection rules cover them. The connection rule (a) is used for the direct connection situation when E_1 outputs flow signals to E_2 and E_2 feedbacks effort signals to E_1. The effort signals are delayed by one period through a one-step-delay block. The connection rule (b) is used in the opposite situation. The connection rule (c) is used for the multiple connections situation when E_1 and E_2 output flow signals to E_3 and E_3 feedbacks effort signals to E_1 and E_2. The connection rule (d) is used in the opposite situation from (c).

3.5 Plant Model Simplification Framework

We assume that the bond-graph plant models, which are configured to approximate the plant behavior. The configuration originally developed for simulation purposes, are reused for verification purposes. These plant models are highly sophisticated because they are designed to check whether the controller model (see Fig. 1) meets functional and real-time requirements such as the increase of a parameter value by a specific amount within specific time. Unfortunately, formal verification approaches are commonly very sensitive to the complexity of the verification target. Luckily, the verification of safety properties can commonly be conducted on dramatically less complex models that overapproximate the original model's properties. However, as we mentioned in Section 3.4, most verification engineers do not have the knowledge to approximate plant models in MBD for verification. As a consequence, the construction of additional simpler models for the sole purpose of verification does not only require redundant work, but it is also error-prone if conducted by the verification engineers who have limited experience in crafting such models. Therefore, we propose an automatic simplification framework, which helps the engineers to conduct simplification for bond-graph plant models to reduce computation load without requiring manual model redesign.

The idea of the simplification approach is to find elements,

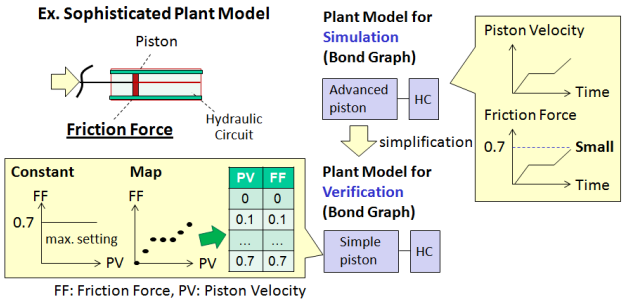


Fig. 12 An overview of simplification approach using an example of sophisticated plant.

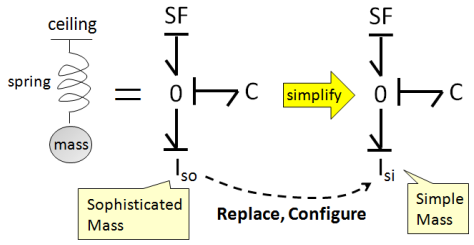


Fig. 13 An example of simplification approach by 1 : 1 element replacement.

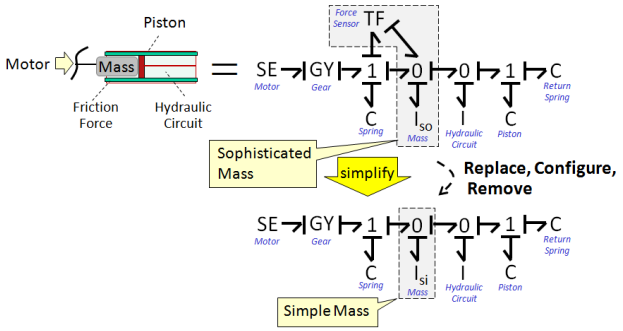


Fig. 14 An example of simplification approach by N : 1 element replacement.

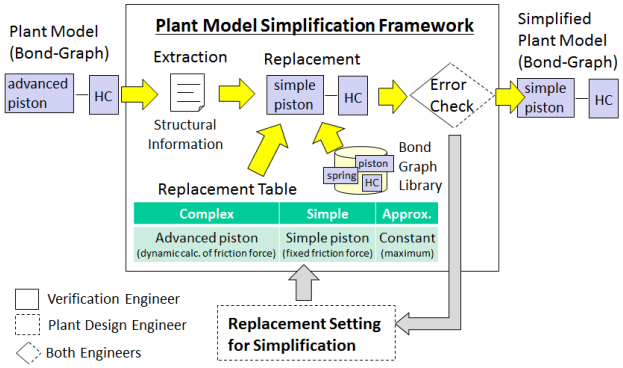


Fig. 15 An overview of the automatic plant model simplification framework.

which produce excessive computational load but have little impact on verification-relevant parameters and replace them with functionally equivalent elements that create lesser computational load during verification. **Figure 12** shows an overview of this simplification approach using an example of a sophisticated plant model. In this example, the plant model for simulation includes an advanced piston, which dynamically calculates friction force according to the current velocity. This dynamic calculation approach can provide highly accurate physical simulation.

Our proposed approach replaces advanced elements, such as the advanced piston, by simple elements, such as the simple piston. Which element should be replaced depends on the target system. For example, in a brake control system the advanced piston is a candidate element for replacement because the physical size of the piston and its amount of produced friction force are relatively small. In this case, a simple piston with a configurable constant friction force parameter is a viable substitute. The parameter is based on simulation results, which are calculated before the replacement procedure. Two examples for such configurations are (1) the constant configuration of a parameter value as its maximum assumed in simulation and (2) the definition of a mapping table for a series of values the parameter assumes in simulation depending on some other parameter. The two options are illustrated in the lower left of Fig. 12. We assume that the replacement setting for an approximation by utilizing element replacement procedure was conducted by plant design engineers.

Figure 13 illustrates the simplification approach by 1 : 1 element replacement. In this example, a sophisticated mass I_{so} of a mass spring model is replaced with a simple mass I_{si}. The left bond graph model is equal to the right simplified mass spring model at the symbolic level. Our proposed simplification method replaces an element by a simpler instance of the same element type such as I_{so} and I_{si} in Fig. 13 and configures the parameter of the replaced element according to fixed parameters of the original elements (such as weight) and variable parameters, which are dynamically calculated. Dynamically changing parameter values are based on simulation results.

Figure 14 shows the simplification approach by N : 1 element replacement. In this case we replace target elements by the same procedure as 1 : 1 element replacement. Furthermore, we remove irrelevant elements. In this brake control system example in Fig. 14, the simplification method replaces a sophisticated mass including a relative element with a simple mass. The velocity of sophisticated mass is measured by leveraging TF, which represents a force sensor to monitor mass force calculated on the basis of mass velocity, and dynamically calculates friction force of the mass according to its velocity. The friction force is used for calculation of precise mass force. For example, if we want to replace a sophisticated mass with a simple mass, the simplification method replaces I_{si} of the above bond-graph model. The method also removes TF as an irrelevant element because the simplified I_{si} uses fixed friction force instead.

Figure 15 shows our proposed automatic plant model simplification framework. The replacement and configuration setting is implemented by plant design engineers as a replacement table and is reused by verification engineers. Our proposed framework conducts automatic simplification according to this table. The replacement table stores information on the relationship of complex elements to their simpler replacement candidates in the target system and information on the recommended configuration approach for variable parameters.

Our framework extracts structural information from a bond-graph plant model stored in an input file. Then, the method conducts element replacement by identifying candidate elements for

replacement in the bond-graph according to the replacement table and structural information, replacing them by simple ones, and setting the simpler elements' configuration parameters. In Fig. 15, the maximum value of dynamic friction force observed in simulation results is used as a constant value of static friction force.

The framework requires the replacement table with replacement setting. We assume the initial setting is implemented as the replacement table by plant design engineers. The verification engineer conducts the simplification procedure according to our proposed framework.

An error check of the simplified model is conducted by both the verification engineer and the plant design engineer in order to evaluate whether the accuracy of the simplified plant model can satisfy the verification purpose. If the accuracy is acceptable, the simplification procedure finishes. If not, e.g., due to over-approximation, the plant design engineer makes a new replacement setting and updates the replacement table. Then the verification engineer conducts the replacement procedure again. The simplification process by using the framework is shown as the following. The framework:

1. reads a plant model from the input file,
2. runs the simulation,
3. reads the replacement table,
4. stores target simulation result according to the approximation option
5. deletes target components of the plant model,
6. adds components and stores instance ID of the added components,
7. connects ports of the added components to one of the other,
8. sets constant value of the added component by the value stored at 4., and
9. runs simulation for error checking.

After the above procedure, the error check was conducted by verification engineers and plant design engineers.

The replacement table consists of component class ID for advanced component (A_CLASS_ID), instance ID of the class ID (A_INS_ID), other component class ID for simple component (S_CLASS_ID), information of approximation option. For example, if the option is approximation by constant, the information consists of target variables of A_CLASS_ID, the referred value of the variable, such as maximum value, and copied variables of S_CLASS_ID. For example, if maximum value is selected, the framework stores the maximum value of the target variable of A_INS_ID of the simulation result.

At step 5, components with A_INS_ID described in the replacement table are deleted. The A_CLASS_ID and A_INS_ID include 0/1 junction or removed element information, such as TF in Fig. 14. At step 6, a component of S_CLASS_ID is added and the instance ID of the simple component (S_INS_ID) is stored. At step 7, ports of the added component are connected with the corresponding ports, which connected to ports of deleted components. The framework has the information on port mapping, which create port connections to make the same energy flow direction with before, between A_CLASS_ID and S_CLASS_ID. At step 8, constant value of the component with S_INS_ID defined at step 6 is set by the value stored at step 4.

4. Experiment: Case Study 1

We conducted a case study on safety verification of a simplified automotive brake control system in order to check the feasibility of our proposed system model construction method and safety verification based on symbolic execution. We attempted to find difficult-to-find malfunctions in an automotive brake control system involving the whole control system. The found malfunction results in faulty unintended braking behavior which was fortunately found during driving test of a commercial car.

We present the experiment environment, the target CPS, verification setting, and verification result in Sections 4.1, 4.2, 4.3 and 4.4 respectively.

4.1 Experiment Environment

For this experiment, we implemented a system model generator from scratch and applied KLEE-MultiSolver [20], which is an extension of KLEE [21], for the symbolic execution based formal verification to implement our proposed verification process. KLEE is well known as a stable practical symbolic execution tool. KLEE-MultiSolver has mechanisms to support the use of different satisfiability modulo theories (SMT) solvers such as STP solver [22] and Z3 solver [23].

In the case study, the control software and the plant model described by ODE were implemented by ourselves as a simplified real-world automotive brake control system. Specifically, we firstly abstracted a specification of the real product and implemented a simplified Simulink plant model and C language control software of two ECUs. Then, we conducted safety verification using system model generation and KLEE-MultiSolver in accordance with given safety requirements. The details of the brake control system are presented in Section 4.2. The verification result is discussed in Section 4.4.

Our experiment ran on a machine with 3.60 GHz Intel® Xeon® quad-core processors and 8 GB of RAM. Operating system is Ubuntu® 12.04 64-bit. KLEE-Multisolver configuration is default. We used the STP solver which is the default solver of KLEE-Multisolver.

4.2 Example of Safety Critical CPS: Brake Control System

Figure 16 shows an overview of the simplified brake control system we used in the first case study. There are two ECUs, a brake control unit (BCU), an electronic stability control unit (ESC), and a brake caliper which is a speed reducer using hydraulic pressure (HP). BCU controls the caliper by producing hy-

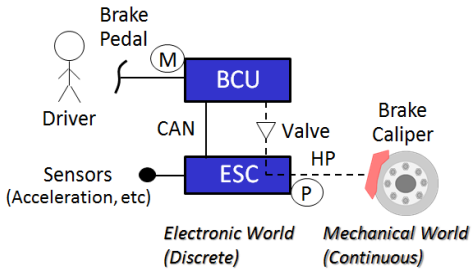


Fig. 16 Brake control system.

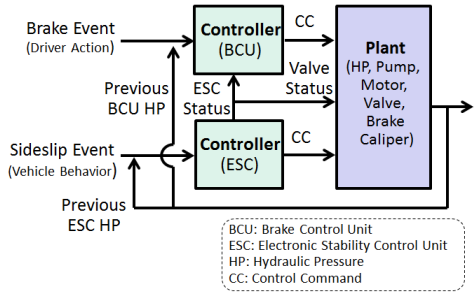


Fig. 17 Brake control system model.

draulic pressure using a brake assist motor in accordance with the brake pedal position. If the car sideslips, ESC takes over hydraulic pressure control of BCU and controls the caliper by producing hydraulic pressure using a pump to stabilize the car. ESC is only active during phases of sideslip. When ESC acts, the valve of the hydraulic pressure circuit is closed by ESC and its pump produces the hydraulic pressure of the ESC side by absorbing oils from the BCU side (the upstream side) of the valve. After the car becomes stable, ESC reopens the valve and returns the hydraulic pressure control to BCU. To avoid the collision of shared brake actuator control between the two ECUs, they coordinate each other using an in-vehicle network called Controller Area Network (CAN) and only one ECU can control the actuator. The interactions between ECUs, and between ECUs and hydraulic pressure control add to the complexity of the system.

Figure 17 shows an overview of the brake control system model. The model is abstracted at high level data communication. This means that the interaction between controller model and plant model is based on control commands, not low level data communication such as pulses to control the brake assist motor and so on. The control command is calculated on the basis of proportion control, which is simplified for the case study, by the control software. Each ECU executes a 1 millisecond periodic task which calculates the current control commands.

The hydraulic pressure, the pump, the brake assist motor, the valve, and the brake caliper were modeled as a plant including 121 blocks in Simulink. The plant code for HILS was generated by Simulink Coder™ at 100 microseconds discretization. These control software, which consist of the controller and basic software that is only a task activator of the operating system because of our abstraction, were manually implemented. We obtained the system model of the brake control system using our implemented system model generator.

4.3 Verification Setting

We present the property and the symbol definition for the verification target in this section.

In order to find the known malfunctions, we defined the nonexistence of unintended brake behavior in the brake control system as a property. This property is one of the most general safety relevant properties in a brake control system. We divided the property into the following conditions which should be always satisfied.

(1) Driver doesn't push brake pedal
(2) Sideslip of the car doesn't occur
(3) Brake force doesn't increase after 500 ms under satisfactions of (1) and (2)

These conditions are implemented as an assertion code. The 500 ms of (3) indicates waiting time for the response delay of the plant model to avoid false positives shown in Fig. 9. While the response time is defined in the plant specification, in the case study, we fixed the waiting time by trial and error. For example, we firstly conducted safety verification of the case study by utilizing 100 ms as the waiting time because we considered that the time need to be set more than control period of the software and to be taken into account response delays of the actuator behaviour. If the verification find false positive, we added further 100 ms to the waiting time and re-verified until no false positive appears. By such process, we fixed the waiting time. We consider that the approach is better than opposite because the long time may cause false negative.

The system level malfunction of the brake control system only appears at the specific combination of the driver's specific brake and the car's specific sideslip given a specific combination of sequence and timing. The brake depends on the brake pedal stroke which means moving distance from the initial position and the sideslip depends on the car's speed. While we should define these analog system inputs as symbols, the available symbolic execution tools cannot deal with floating point data types for analog data expression in Simulink. For example, Ariadne can deal with floating point data types, but is not published yet [24]. Consequently, we transformed the analog system inputs into binary system inputs such as brake occurrence and sideslip occurrence. For example, the brake occurrence is expressed as ON or OFF. ON means to strongly push the brake pedal like sudden brake. OFF means to release the brake pedal. In the case of the sideslip occurrence, ON means that the car sideslips at high speed and OFF means that the car is stable. This approach enables the available tools to define analog system inputs as symbols indirectly.

Additionally, to find the malfunctions given a specific combination of sequence and timing, we used iterative symbol definitions of respective system inputs. As the optimal redefinition frequency depends on verification targets, the frequency was clarified by trial and error. The verification time depends on the redefinition frequency because we need to define significant time to check the system behavior affected by the system input's changes. In the case study, the verification time is tentatively limited at 5 seconds taking into account the response delay of the plant behavior. Furthermore, due to efficient verification on conditions of overlapping system inputs, we inserted time offsets into the timing of the brake occurrence.

The system model generator generated the system model which consists of the control software of BCU and ESC, the plant code, the communication module, the symbol definition module, the assertion module, and the synchronization module. The system model is approximately 2,000 code lines of C language source code.

4.4 Verification Results

To find the complex system level malfunction, we tried to conduct the safety verification of the brake control system with dif-

Table 1 Symbol definition frequency and malfunction.

symbol definition frequency	malfunction detection	execution time
every 100 us	no (within one day)	-
every 100 ms	no (within one day)	-
every 1 second	yes	9 hours 9 minutes
every 2 seconds	yes	31 minutes

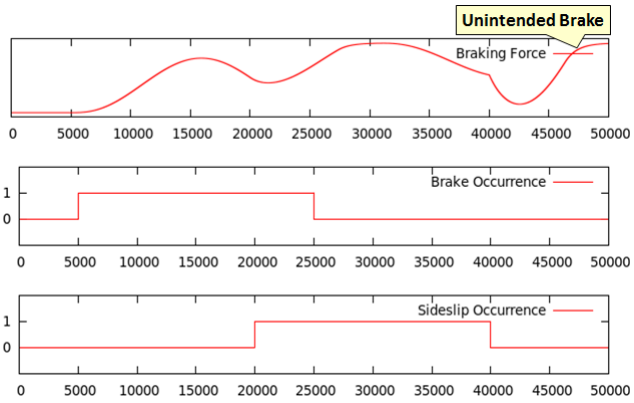


Fig. 18 Brake force behavior in Unsafe Case.

ferent symbol definition frequency. Finally, the safety verification detected the expected malfunction.

Table 1 shows relationships between the symbol definition frequency and the malfunction detection. While the verification with the symbol definition of short time duration could not finish within one day, in the case of long time duration such as a second or 2 seconds, the verification finally could find the complex malfunction. The duration of the second time scale satisfies practical demands because our target CPS behavior is changed by human operation events or environment event generated at second time scale mentioned in Section 2.1. As the result, the symbol definition works for system input calculation of the human operation event or environment event in the target CPS. However, if one needs to apply our approach to other CPSs, such as the system operated by electronic control event, the definition frequency would not be sufficient because the required frequency of system input generation is millisecond scale.

To understand the details of the system behavior, we conducted a simulation using the system input pattern resulting in the property violation. **Figure 18** shows the simulation result. As the graph shows, the car's sideslip occurs during sudden braking and then once the car becomes stable, unintended brake force appears. This is the complex system level malfunction given by a specific combination of sequence and timing.

Figure 19 shows the details of the system behavior in the error case. BCU involves a diagnostic program which detects oil leak on the basis of gaps between pedal stroke value and the amount of hydraulic pressure. As the graph shows, after the stability control of ESC, there exists a big gap because the amount of the oil to stabilize the high speed car was more than the prediction. Therefore, the diagnostic program detected the oil leak due to the gap which exceeds the error threshold and then BCU invoked a fail-safe program which automatically produces hydraulic pressure by making the brake assist motor pull the brake pedal in order to brake by the rest of the oil. As the result, unintended brake

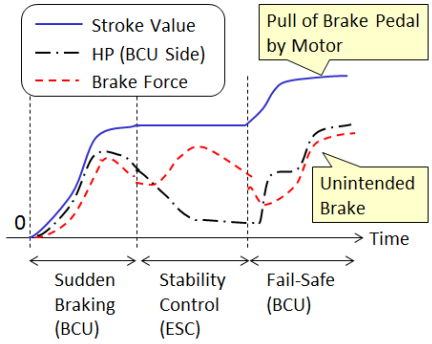


Fig. 19 Details of system behavior in error case.

occurs.

The factor of the system level malfunction is related to gaps between response delays of the ECU processing (digital) and the hydraulic pressure behavior (analog). While a conventional top-down system development process of safety critical CPS bears the potential to cause the difficult-to-find system level malfunctions, through this case study, we established that our proposed safety verification approach can detect them.

5. Experiment: Case Study 2

In order to evaluate our proposed plant model simplification framework and transformation method, we conducted the second case study on the safety verification of a brake control system, which contain bond-graph plant model described by DAE.

In this section, we present our experiment environment for the second case study in Section 5.1, the obtained experimental results, and a discussion about how to interpret them in Section 5.2.

5.1 Experiment Environment

To validate the feasibility of the proposed methods, three experiments have been conducted in the second case study according to the verification process shown in Fig. 7. In the first experiment of the second case study, we measured how much our model simplification framework reduces computation load with tolerating any decrease of verification accuracy. While the contribution of our proposed simplification framework supports the plant model simplification procedure by automation, the framework restricts the simplification approach for computation load reduction. In the second experiment, we measured to which degree our model transformation method introduces behavioral deviations between the bond-graph model and the signal-flow model. In the third experiment, we measured time for conducting automated safety verification of the transformed model and checked whether the verification enable failure detection after our proposed transformation and simplification method. As a verification tool, we chose KLEE with default configuration of the symbolic execution engine and the STP solver for solving path constraints. In the first and second experiments, the operating system is Windows® 7 32-bit and in the third experiment it is Ubuntu® 12.04 32-bit. The first and second experiments ran on a machine with 2.80 GHz Intel® Core i5 quad-core processors and 2 GB of RAM and in the third experiment it with 3.0 GHz Intel® Core i7 quad-core processors and 16 GB of RAM.

We applied the above experiments to an automotive brake con-

trol CPS. We implemented our proposed plant model simplification framework and transformation method as two prototype tools. For the model transformation tool, the insertion of delay blocks has been conducted manually. Additionally, in the second experiment the model generation procedure, which is shown in Fig. 10, is also conducted manually, in the sense that signal-flow subsystems for model transformation were developed manually before the model transformation tool execution. The tool makes use of these manually developed subsystems in the component placement procedure shown in Fig. 10. As the manually identified (but automatically applied) subsystems are reusable for other models that contain the same subsystems, we consider this a one-time effort. For example, since the signal-flow piston subsystem is frequently used in many plant models of the same domain, we consider the subsystem replacements to be reusable at least for the same domain, i.e., the product family.

For these experiments, we developed a brake control system model of the target CPS, shown in Fig. 2, which consists of a controller model implemented in C, and a plant model designed in AMESim. The plant model is discretized using 100 micro seconds intervals. This means that one-step delay blocks delay the target signals for 100 micro seconds. In the first experiment we measured the effect assessed by simulation and reused the plant model of the real mass production development with the same discrete time intervals instead of our developed model in order to measure the accurate effect.

We embedded a subtle fault in the control software, leading to an unintended brake force. In the third experiment, the verification must detect this fault by checking the resulting violation of the safety requirement, i.e., unintended braking does not occur. We implemented three conditions to detect the safety requirement violation as assertion code. If all of these conditions are satisfied, the safety requirement is violated. The first condition is that the brake pedal is not actuated. The second condition is that the elapsed time is at least 500 ms after the pedal released. The time prevents the verification tool from misdetections caused by the response delay of the plant. The third condition is that the amount of piston displacement, which means distance from initial piston position, increases, i.e., the brake force increases. A brake pedal operation, which is a system input, is defined as a symbol supplied by KLEE every 1 second through symbol re-definition. We generated the system model combined by the control software, the plant code, and the assertion code. The model replicates the system behavior during 5 seconds in verification time and is structured by 2,000 lines in C. We verified the safety of the model.

5.2 Experiment Results

Figure 20, **Table 2**, and **Fig. 21** show the first experimental results of the plant model simplification method. The simplification tool found an advanced mass and replaced it to simple mass and configured it for approximation according the replacement table (see Fig. 20). This single replacement yields approximately 35% computation load reduction in simulation (see Table 2).

Additionally, in order to compare the simplified plant model behavior with the original one, we plot their behaviors in Fig. 21 for an easy visual comparison. As the result shows, there is no

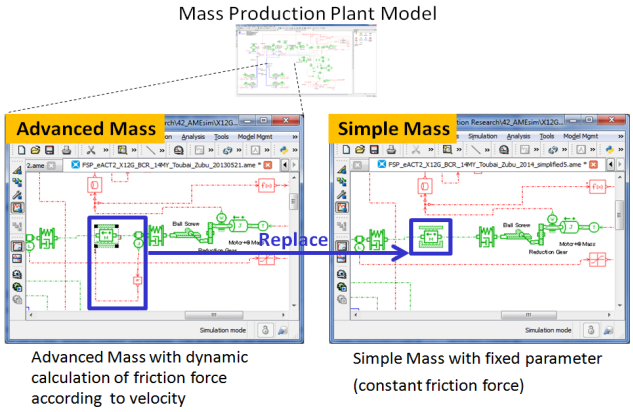


Fig. 20 The first experimental result on the model simplification framework.

Table 2 Comparison results of simulation time.

Plant Model	Average of Simulation Time [s]	Reduced Rate [%]
before simplification (advanced piston)	179	-
after simplification (simple piston)	116	35.2

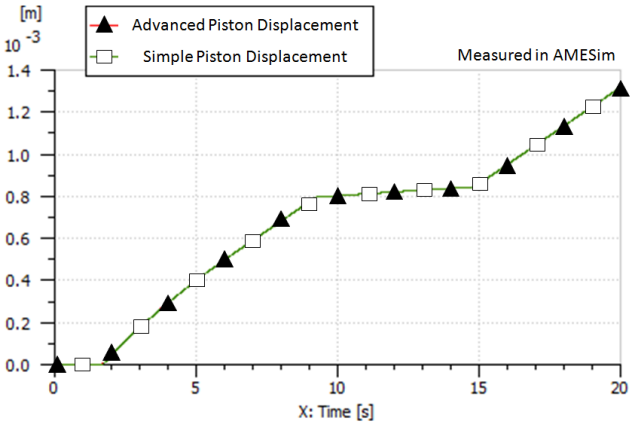


Fig. 21 Comparison results of plant behavior before and after simplification.

recognizable difference. This means our model simplification framework can supply the simplification without affecting the accuracy of the model although the simplification results depend on the replacement setting done by the plant design engineer.

Figure 22 shows the second experimental result on the model transformation method. The model transformation tool could transform the bond-graph plant model of the brake control system described in AMESim into a signal-flow plant model for MATLAB/Simulink. The plotted simulation result shows the brake piston displacement of the original model (left) and the transformed model (right) with changing brake force. To achieve a sound comparison, we applied the same settings for the numeric solver method and configuration parameter in both AMESim and MATLAB/Simulink. As the result in Fig. 22 illustrates, there was no recognizable error. **Figure 23** shows the third experimental result on the safety verification. In the upper part of the figure illustrating the driver brake pedal operation, ON indicates pedal actuation. OFF indicates pedal release. The piston displacement in the lower part of the figure shows that there is an increase (in-

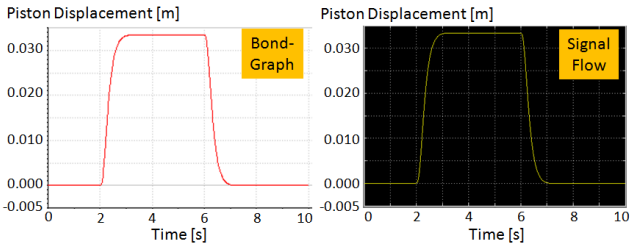


Fig. 22 The second experimental result on the model transformation method.

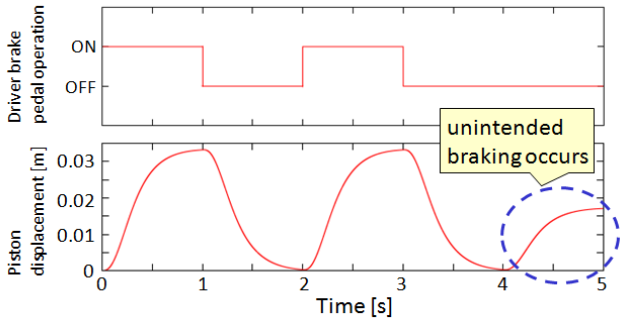


Fig. 23 The third experimental result on the safety verification.

dicating brake engagement) despite pedal release between seconds 4 and 5. This means that unintended braking occurred even though the driver did not actuate the brake pedal. This failure was detected by spending 23 minutes 15 seconds of execution time. This failure condition is the expected result of the fault we embedded. Therefore, the result confirms that the verification tool was able to properly detect the fault that represents a potential safety threat from the system model including the plant model transformed by our prototype tool. After the failure was fixed, we conducted the safety verification. The safety was verified by spending 23 minutes 44 seconds of execution time.

6. Discussion

In order to facilitate the automated verification of safety properties in critical CPSs, we have proposed model transformations and simplifications. Obviously, none of the proposed modifications to the CPS models should affect the validity of the verification. In the following we discuss the effects of our modifications on the system behavior.

6.1 Effects from the Model Transformation Approach

Our model transformation approach inserts one-step delay blocks between components. In order to clarify the impact caused by the blocks, we measured each output signal value of the transformed plant model in Simulink in the case of one-step delay, which our approach inserts, and in the case of two-step delay for the comparison to extract the difference. We did not measure the signal value of the model with no delay blocks because the model did not work in Simulink due to the algebraic loops. As we mentioned earlier, one step means 100 micro seconds. The measurement was done at the same condition of the second experiment in Simulink. The result showed that the maximum difference of plant output signal was 0.1%. We consider that the difference is acceptable because it's smaller than the modelling error.

Bond-graph modeling enables us to design the plant model by connecting physical components. Our transformation method inserts a delay block into a feedback loop between two components. The block delays one simulation step such as 100 micro seconds to the value of the feedback. This means each component state is individually updated by the current input signal and the previous output signal as reaction value. Therefore, the delays caused by the blocks do not sum up. Consequently, the effect from our model transformation approach is acceptable.

6.2 Effects from the Model Simplification Approach

The purpose of our safety verification method is to check software logical error resulting in the safety violation from the view of the system behavior. In order to reduce the verification complexity of the plant model, our model simplification approach abstracts the target plant model. The model abstraction approach may cause false positives or false negative. Therefore, as our simplification framework relies on the replacement setting defined by plant design engineer, the effects of the simplification must be evaluated by verification engineers and plant design engineers at the error check step in Fig. 15. While our proposed framework can support the automation of the simplification procedure, the automation of the error check procedure is a challenge for the future work.

The simplification framework limits approximation method because the replacement procedure works in situations where bond-graph tools have simple component models and their ports complying with the ports of advanced component models. The experiment result in Section 5.2 showed that the limitation is acceptable.

7. Related Work

There are related work for multi-domain model-based design and verification approach [25], [26]. The approach enables the safety verification of a system model made up by hybrid automata [27]. The system model is individually created by users without utilizing controller models or plant models as verification view of the system design although the system model imports constraints between each model behavior defined in heterogenous models [28], [29]. As described in Section 1, it's difficult for industry to create hybrid system models described in hybrid automata from scratch. Our approach applied an automatic generation of system models utilizing MBD as described in Section 3.2. Consequently, our approach is practicable if developed software and plant models are ready at the phase of the safety verification. As a similar approach, Crescendo tools are provided by DESTECS project [30]. The project also presents synchronization mechanism, called co-models, between software and plant models [31]. However, the co-models do not support the limitation of the verification time because the project focuses on the simulation.

Simulink Design Verifier [32] has the capability of verifying the Simulink model. It can verify the functionality of the control system. However, if failures of the verification target are embedded into basic software such as I/O driver described in C, the verification cannot detect the failure because the system model of the verification target doesn't include the basic software. Our

proposed verification process has the capability of verifying the system model including the basic software. Therefore, our process is suitable for the safety verification.

Majumdar et al. [33] present a symbolic execution technique for closed-loop control systems. The technique realizes robustness verification, which verifies system control stability. This research doesn't discuss about how to cope with the response delay of plant models for property definition, which is described in Section 3.3.

8. Conclusion

In this paper we propose a practical verification process for the safety verification of safety critical CPSs. The process conducts the system model construction, which automatically generates system models of the verification target, and the safety verification of the system model in accordance with safety relevant properties by symbolic execution based formal verification. Our approach has an advantage in comparison with simulation approach because simulation approaches require the manual specification of test inputs causing system level malfunctions. In contrast, the proposed approach identifies possible violations of safety properties from model analyses along with the input conditions under which the violations occur.

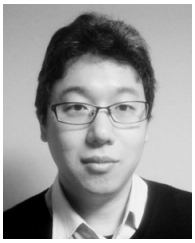
Furthermore, the verification process conducts the plant model transformation and simplification for verification complexity reduction. The developed model transformation method transforms bond-graph plant models with algebraic loops into signal-flow models without algebraic loops to make them applicable for existing automated verification approaches. The model simplification framework support the plant model simplification procedure, which replaces complex components by simpler ones that exhibit approximate behavior to a sufficient degree, by automation of a part of the procedure. In the framework, the replacement is based on expert knowledge, which is captured in replacement libraries for reusability, and application-specific parameter tuning based on simulation. We applied the proposed verification process for two case studies on the safety verification of a safety-critical automotive brake control CPS. The first experiment results showed that our verification approach can detect difficult-to-find system level malfunctions from the abstracted system model by iterative symbol and property definitions taking into account the response delay of the plant behavior. The second experimental results showed that the model simplification framework yields approximately 35% computation load reduction in simulation and the model transformation method yields signal-flow models without recognizable errors. Additionally, the proposed verification approach was able to correctly detect unsafe behavior of the brake control system model, which was transformed. In future work, we plan to develop an error localization method to aid debugging when safety violations are indicated by the presented verification approach.

Acknowledgments Research supported in part by TUD CySEC. We also thank Hitachi Automotive Systems for providing the application examples.

References

[1] ACATECH (Ed.): Cyber-Physical Systems - Driving Force for Innovation in Mobility, Health, Energy and Production (2011).
[2] Lee, E.A. and Seshia, S.A.: *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*, available from ⟨http://LeeSeshia.org⟩ (2011).
[3] Alur, R.: Formal Verification of Hybrid Systems, *Proc. ACM Intl. Conf. Embedded Software*, pp.273–278 (2011).
[4] Henzinger, T.A., Ho, P. and Wong-toi, H.: HyTech: A model checker for hybrid systems, *Proc. ACM Intl. Conf. Computer Aided Verification*, pp.460–463 (1997).
[5] Frehse, G.: PHAVer: Algorithmic Verification of Hybrid Systems past Hytech, *Journal on Software Tools for Technology Transfer*, Vol.10, Issue 3, pp.263–279 (2008).
[6] Tiwari, A.: HybridSAL Relational Abstracter, *Proc. ACM Intl. Conf. Computer Aided Verification*, pp.725–731 (2012).
[7] Platzer, A. and Quesel, J.: KeYmaera: A Hybrid Theorem Prover for Hybrid Systems, *Proc. Intl. Conf. Automated Reasoning*, pp.171–178 (2008).
[8] Lerda, F., Kapinski J., Maka, H., Clarke, E.M. and Krogh, B.H.: Model Checking In-The-Loop: Finding Counterexamples by Systematic Simulation, *Proc. IEEE American Control Conference*, pp.2734–2740 (2008).
[9] Kawahara, R., Dotan, D. and Sakairi, T.: Verification of embedded system's specification using collaborative simulation of SysML and simulink models, *Proc. IEEE Intl. Conf. Model-Based Systems Engineering*, pp.21–28 (2009).
[10] Nakajima, S., Furukawa, S. and Ueda, Y.: Co-Analysis of SysML and Simulink Models for Cyber-Physical Systems Design, *Proc. IEEE Intl. Conf. Embedded and Real-Time Computing Systems and Application*, pp.473–478 (2012).
[11] Coen-Porisini, A., Denaro, G., Ghezzi, C. and Pezze, M.: Using Symbolic Execution for Verifying Safety-Critical Systems, *Proc. ACM Intl. Conf. European Software Engineering*, pp.142–151 (2001).
[12] Ishigooka, T., Saissi, H., Piper, T., Winter, S. and Suri, N.: Practical Use of Formal Verification for Safety Critical Cyber-Physical Systems: A Case Study, *Proc. IEEE Intl. Conf. Cyber-Physical Systems, Networks, and Applications*, pp.7–12 (2014).
[13] Ishigooka, T., Saissi H., Piper T., Winter, S. and Suri, N.: Practical Formal Verification for Model Based Development of Cyber-Physical Systems, *Proc. IEEE/IFIP Intl. Conf. Embedded and Ubiquitous Computing*, pp.1–8 (2016).
[14] The MathWorks Inc.: Simulink R2016b (online), available from ⟨http://www.mathworks.com/products/simulink/⟩ (accessed 2016-11-08).
[15] Siemens PLM Software: Amesim (online), available from ⟨http://www.plm.automation.siemens.com/en_us/products/lms/imagine-lab/amesim/⟩ (accessed 2016-11-08).
[16] The MathWorks Inc.: Simscape R2016b (online), available from ⟨http://www.mathworks.com/products/simscape/⟩ (accessed 2016-11-08).
[17] The Modelica Association: The Modelica Specification, Version 3.3 Revision 1 (online), available from ⟨https://www.modelica.org/⟩ (accessed 2016-11-08).
[18] Marks II, R.J.: *Introduction to Shannon Sampling and Interpolation Theory*, Springer-Verlag (1991).
[19] Borutzky, W.: Bond Graph Modelling and Simulation of Mechatronic Systems An Introduction into the Methodology, *Proc. European Conf. Modeling and Simulation* (2006).
[20] Palikareva, H. and Cadar, C.: Multi-solver support in symbolic execution, *Proc. ACM Intl. Conf. Computer Aided Verification*, pp.53–68 (2013).
[21] Cadar, C., Dunbar, D. and Engler, D.: KLEE: Unassisted and Automatic Generation High-Coverage Tests for Complex Systems Programs, *Proc. USENIX Conf. Operating Systems Design and Implementation*, pp.209–224 (2008).
[22] Ganesh, V. and Dill, D.L.: A Decision Procedure for Bit-Vectors and Arrays, *Proc. Intl. Conf. Computer Aided Verification*, pp.519–531 (2007).
[23] Moura, L.D. and Bjorner, N.: Z3: An efficient SMT solver, *Proc. ACM Intl. Conf. Tools and Algorithms for the Construction and Analysis Systems*, pp.337–340 (2008).
[24] Barr, E.T., Vo, T., Le, V. and Su, Z.: Automatic Detection of Floating-Point Exceptions, *Proc. ACM Symposium on Principles of Programming Languages*, pp.549–560 (2013).
[25] Rajhans, A. and Krogh. B.H.: Heterogenous Verification of Cyber-Physical Systems using Behavior Relations, *Proc. ACM Intl. Conf. Hybrid System Computation and Control*, pp.35–44 (2012).
[26] Bhave, A., Krogh, B., Garlan, D. and Schmerl, B.: Multi-domain Modeling of Cyber-Physical Systems Using Architectural Views,

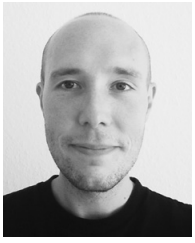
Proc. Analytic Virtual Integration of Cyber-Physical Systems Workshop (2010).
[27] Henzinger, T.A.: The theory of hybrid automata, *Proc. IEEE Symposium on Logic in Computer Science*, p.278 (1996).
[28] Bhave, A., Krogh, B.H., Garlan, D. and Schmerl, B.: View Consistency in Architectures for Cyber-Physical Systems, *Proc. IEEE/ACM Intl. Conf. Cyber-Physical Systems*, pp.151–160 (2011).
[29] Rajhans, A., Bhave, A., Loos, S., Krogh, B.H., Platzer, A. and Garlan, D.: Using Parameters in Architectural Views to Support Heterogeneous Design and Verification, *Proc. IEEE Conf. Decision and Control and European Control*, pp.2705–2710 (2011)
[30] DESTECS project: The Crescendo Tool (online), available from ⟨http://crescendotool.org/⟩ (accessed 2017-03-01).
[31] Fitzgerald, J., Pierce, K. and Larsen, P.G.: Co-modelling and Co-simulation in the Engineering of Systems of Cyber-physical Systems, *Proc. IEEE Intl. Conf. System of Systems Engineering*, pp.67–72 (2014)
[32] The MathWorks Inc.: Simulink Design Verifier R2016b (online), available from ⟨https://jp.mathworks.com/products/sldesignverifier/⟩ (accessed 2016-11-08).
[33] Majumdar, R., Saha, I., Shashidhar, K.C. and Wang, Z.: CLSE: Closed-Loop Symbolic Execution, *Proc. Intl. Symposium NASA Formal Methods*, pp.356–370 (2012).



Tasuku Ishigooka received a master degree from Musashi Institute of Technology in 2008. He is a researcher at Center of Technology Innovation - Controls, Research and Development Group, Hitachi Ltd. His research topics include real-time distributing processing design and verification.

Habib Saissi is a Ph.D. student at TU Darmstadt. His work targets the development of formal methods for the efficient verification of software systems.

Thorsten Piper received his Ph.D. from TU Darmstadt in 2015. His research targets the assessment and design of safety mechanisms for safety-critical software systems. He is currently with Continental Automotive.



Stefan Winter has obtained a doctoral degree in Computer Science from TU Darmstadt in 2015, where he is now working as a postdoctoral research fellow. His research focuses on the design and analysis of dependable software systems.

Neeraj Suri received his Ph.D. from the UMass-Amherst and is a Chair Professor at TU Darmstadt, Germany. His research addresses the design, analysis and assessment of trustworthy systems and software.