Modelling and Certification for Electric Mobility

Alexander Graf-Brill*, Arnd Hartmanns†, Holger Hermanns* and Steffen Rose‡
*Saarland University, Saarbrücken, Germany
†University of Twente, Enschede, The Netherlands
‡emtas GmbH, Merseburg, Germany

Abstract—**The ENERGYBUS specification is the basis of an ongoing joint IEC/ISO standardisation effort focussing on public charging infrastructures for and interoperability of light electric vehicle components. This paper highlights how these efforts are supported by formal methods, starting at the design and specification level, up to establishing a certification framework for standards compliance of devices implementing the specification. The MODEST TOOLSET supports the model-based analysis methods needed in this context.**

I. INTRODUCTION

Embedded control software has become a major driver of industrial innovation, encompassing many critical, and sometimes safety-critical, application domains. A particularly delicate domain is the management of electric power: Embedded power management software has been traced to be the root of unintended and partly dangerous malfunctionings of laptops [1], smart phones [2], smart watches [3], pacemakers [4], and light electric vehicles [5]. The proper handling of electric power by software is obviously intricate. At the same time, electric power is the basic commodity needed to innovate formerly all-mechanical systems such as bicycles. Indeed, many embedded innovations are being developed in this domain, with pedelecs and e-bikes being the most prominent examples.

ENERGYBUS is an open specification for interoperability of the electrical components of e-bikes and other light electric vehicles, encompassing batteries, chargers, motors, sensors, and the human interface. It is driven by EnergyBus e.V., an association formed by major industrial stakeholders in the e-bike domain. The ENERGYBUS specification is the nucleus for the joint IEC/ISO standardisation IEC/IS/TC69/JPT61851-3, aiming at eventually enabling a single charger to be used across all light electric vehicles. By mid 2018, this safety standard will become binding, so as to enable effective public charging infrastructures for light electric mobility.

This paper reviews how state-of-the-art formal methods and tools have been and are being applied in this context to assure the general correctness and safety of ENERGYBUS protocol specifications, as well as to support implementers of ENERGYBUS in designing correct and safe implementations. For the latter, we have developed a tool platform for automated conformance testing of ENERGYBUS implementations against their formal specification. We have made this platform available to the industrial members of EnergyBus e.V. The tool platform is based on the MODEST modelling language [6] and its accompanying MODEST TOOLSET [7], which has been extended with a component supporting effective model-based testing against a formal ENERGYBUS protocol specification.

This paper first reviews the MODEST approach to formal modelling and its features, followed by an explanation of the core principles of ENERGYBUS. Particular emphasis is put on the certification process for ENERGYBUS products, which hinges on the conformance test plan designed by the consortium. We then explain the role of model-based testing in this process and review results achieved in its application.

II. FORMAL MODELLING

Modelling is a first important step to specify the behaviour of implementations operating in complex contexts like ENERGYBUS. More generally, models may be based on an abstract system design, a specification of (un)desired events, or an actual implementation. Suitable modelling formalisms are:

Formal: To obtain trustworthy results that are consistent for different tool implementations of the same formalism, it must be equipped with a complete mathematical semantics.

Behavioural: Embedded software is reactive, i.e. it is continuously operative, acting upon stimuli and generating control signals. We are thus interested in the overall system's behaviour over time, instead of the classic perspective of looking at the input-output function of an algorithm.

Compositional: Building complex systems monolithically does not scale, and the same holds for modelling. We need the ability to reflect the system's component structure in the model in a natural way. Providing reusable model components also makes for efficient and agile modelling.

Quantitative: Functional correctness is necessary, but far from sufficient to consider a design or implementation "correct". For example, performance requirements may prescribe a minimum throughput, availability may be evaluated based on the mean time between failures, and energy consumption must be balanced against a power budget. We thus need to be able to reason about probabilities, time, and continuous physical quantities where necessary.

In practice, it is crucial to pick the right level of detail and abstraction. The variety of quantitative aspects and the inevitable tradeoff between expressivity of formalisms and feasibility of analysis has lead to the development of an entire zoo of formal quantitative modelling formalisms [8].

A. *The Modest Approach*

The MODEST language is a modelling formalism with high-level features like parallel composition, recursive process specifications, abstract datatypes, and exception handling. It has a formal semantics in terms of networks of stochastic

```
 1 process Battery() { // linear battery model
 2   real load = 10; // battery load, initially charging
 3   var soc = 50; der(soc) = load; // state of charge
 4   do {
 5   :: assist_low? {= load = -1 =} // switch to low mode
 6   :: assist_high? {= load = -2 =} // switch to high mode
 7   :: charge? {= load = 10 =} // plugged in, charging
 8   :: when urgent(soc == 100) full! {= load = 0 =}
 9   :: when urgent(soc == 0) empty! }
10 }
11 process Cyclist() { // abstract model of a cyclist
12   process Home() { full?; // wait for full charge
13                    Trip(DiscreteUniform(5, 20)) }
14   process Trip(int km) {
15     clock c = 0; real x = Uniform(2, 10);
16     if(km == 0) { charge!; Home() }
17     else { alt { :: assist_low! :: assist_high! };
18            when urgent(c == x) Trip(km - 1) } }
19   Home()
20 }
21 par { :: Battery() :: Cyclist() } // parallel composition
```

Figure 1. An example of a stochastic hybrid MODEST model

hybrid automata (SHA, [6]). Figure 1 presents an exemplary MODEST model, representing the interaction between a cyclist and the state of charge of their e-bike's battery. The example is made to concisely present the various features of MODEST in a small and abstract manner. It consists of two processes running in parallel (line 21), which communicate via a number of shared actions (`assist_low`, `full` etc.). The cyclist waits at home until the battery is fully charged (line 12), then goes on a trip of between 5 and 20 km (line 13). At the end of each trip, the battery is plugged back in for charging (line 16). The `Battery` process is a simple linear battery model (line 3) that reacts on signals from the cyclist to update its current load (lines 5 to 7) and generates signals itself when the battery is full or empty (lines 8 and 9). This example illustrates the main features of MODEST and SHA:

Nondeterminism: Every kilometre, the cyclist *nondeterministically* chooses whether to continue in low or high assistance mode (line 17). The model thus remains abstract w.r.t. the actual choices of the cyclist, and an analysis should deliver safe worst- and best-case bounds. In general, nondeterminism is *unquantified uncertainty*. It is crucial for compositionality and enables abstraction, concurrency, and underspecification. SHA are thus an extension of Kripke structures or labelled transition systems.

Stochastic choices: Each trip's length is specified *stochastically*: any number of kilometres between 5 and 20 is equally likely (line 13), and the value of variable `x` is sampled anew from the continuous uniform distribution over $[2, 10]$ for every kilometre. This ability to represent stochastic decisions, i.e. *quantified uncertainty*, allows us to model any behaviour where the probability of each option is known. Application examples include randomised algorithms (by specification) or the time between faults (by measurements and statistics). SHA are thus an extension of Markov chains, Markov decision processes, and stochastic automata [9].

Time: Cycling one kilometre takes `x` time units (line 18). In MODEST, real-time aspects are modelled using *clocks* in the same way as in timed automata [10]. We can thus create hard real-time and, by combining clocks and stochastic features as in the example, soft real-time models.

Continuous dynamics: As an extension of hybrid automata [11], MODEST/SHA allow the description of continuous system dynamics by differential equations. Our example uses this for the linear battery model in line 3: the derivative of the state of charge is the load on the battery.

B. Measures of Interest

Formal modelling often uncovers inconsistencies or omissions early in the design process [12]. Thus the act of modelling in itself already leads to an improved design that is much better understood. Yet in the end, the goal is to make the model amenable to analysis so as to enable verification of certain statements and computation of quantities of interest. These may include functional safety requirements, e.g. "the battery will never become empty" in our example (which we would find to be violated because it may happen with *some* positive probability after sufficiently many trips). On quantitative models, typical requirements are of the following forms:

Probabilistic reachability: Basic probabilistic questions are of the type "what is the probability of the battery becoming empty?" However, since our example is nondeterministic, we actually need to ask for minimum (i.e. the "best case" where the cyclist chooses assistance levels such that battery lifetime is long) or maximum probabilities ("worst-case" decisions). Both probabilities are 1, but if we instead ask time-bounded questions—"what is the probability of an empty battery in $\leq t$ time units"—we get nontrivial results.

Reward-based properties: We may add rewards (or costs) to certain behaviours, e.g. assign an instantaneous reward of 1 to completing a trip, or a rate reward of the energy consumption over time. We can then ask reward-bounded probabilistic reachability queries: "What is the probability of an empty battery in the first n trips?" Alternatively, we may compute accumulated reward expectations, e.g. the maximum expected energy consumption or the minimum expected number of trips before the battery is emptied.

Temporal logics: To specify more complex requirements related to the relative ordering and timing of certain signals, quantitative extensions of LTL, CTL or other temporal logics can be used.

C. Domain-Specific Frontends and Interoperability

MODEST models can be converted to JANI [13], an interchange format for quantitative models designed to improve tool interoperability and comparison. It is supported by several analysis tools that implement conversions from various higher-level or domain-specific languages to JANI. Notably, a stochastic-nondeterministic extension of the scenario-aware dataflow formalism that is popular in embedded signal processing applications, called xSADF [14], is supported by the MODEST TOOLSET. For use at Philips, the domain-specific language iDSL [15] was developed for performance evaluation of medical imaging devices, which is transformed to MODEST in order to access the MODEST TOOLSET's analysis backends.

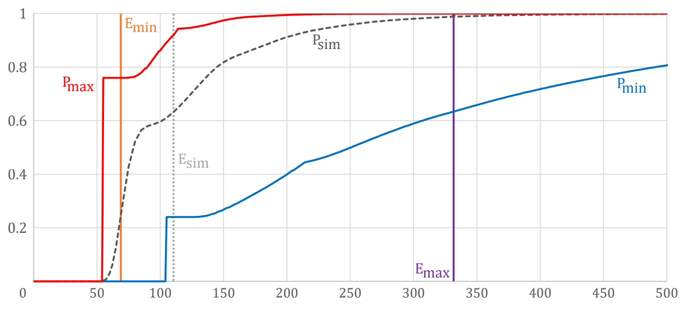

Figure 2. Simulation and model checking results for the example model

III. Model-Based Analysis

Using a quantitative formal model as specified above, there are three complementary avenues for analysis: We can use simulation to better understand the model and refute certain properties, or model checking to verify requirements resp. compute optimal values for quantities of interest. Once we are certain that the design is "good" and properly represented by the model, we can use it as a specification in itself and validate that implementations conform to the behaviour prescribed by the model via model-based testing.

A. Simulation

Simulating the model means generating one particular trace of its behaviour. If the model is stochastic, such a trace is picked randomly and we can use Monte Carlo simulation: generate a large number of traces to compute estimates of probabilities and expected values with some statistical confidence. When using a formal model, this is also called *statistical model checking* (SMC, [16], [17]). In the Modest Toolset, it is supported by the modes simulator and has, for example, been successfully applied to study the behaviour of novel distributed control algorithms for photovoltaic microgenerators [18].

Nondeterminism, however, cannot be simulated: In a nondeterministic choice, there is no information about the frequency with which the available options shall be selected, so it cannot be resolved by randomisation without introducing a (hidden) assumption. If we do so anyway, we will get a single estimate for e.g. a reachability probability that is only guaranteed to lie *somewhere* between the maximum and minimum probabilities that we are actually interested in.

For (a discretised version of) our example, we show simulation results as part of Figure 2. We plot the probability of the battery becoming empty within 0 to 500 time units, with the simulation result P_{sim} as the dashed grey curve, based on a (hidden) random resolution of the cyclist's nondeterministic choices. The vertical dotted grey line is E_{sim}, the simulation estimate for the expected time until the battery is empty.

B. Model Checking

In order to compute precise values for maximum and minimum probabilities or expected values, we need *probabilistic model checking*. It explores the state space of all reachable configurations of the model, then numerically computes the value of interest over the state space using methods like linear programming or value iteration. While model checking naturally handles nondeterminism, it faces the *state space explosion* problem: the number of configurations is exponential in the number of model variables and their domains, so model checking quickly runs out of memory on complex examples.

In the Modest Toolset, model checking is provided by the mcsta tool. It has notably been used to formally verify the safety of a wireless bike braking system [19]—a setting where trustworthy worst-case results were required. We have also used mcsta on our example to compute P_{min} and P_{max}, the actual minimum and maximum probabilities for the battery to become empty over time, plus the corresponding expectations E_{min} and E_{max} as shown in Figure 2.

C. Model-Based Testing

Using a formal model as the specification of desired behaviour, *model-based testing* [20] generates and rolls out a suitable set of experiments in an automated manner on the implementation under test (IUT). The goal is to assert some notion of conformance of the IUT with respect to the model. Model-based testing is usually specified for modelling formalisms that are variants of input-output transitions systems (IOTS) where the transitions between states have structured action labels: the name of a performed action and an identifier for the type of action, i.e. input (stimuli) to the implementation or output (response) of the implementation. The behaviour of every IUT is assumed to be expressible by an IOTS as well.

A model-based testing tool performs automated inspection of the possible inputs and outputs while stepping through the states of a model. In each state it either provides an input to or records an output from the IUT and accordingly updates its knowledge of what the current state of the model is. Whenever an unexpected output of the IUT occurs, i.e. an output which is not foreseen by the current knowledge of the model state, the IUT is refuted with the verdict "fail". Formally, the basic model-based testing process consists of several components:

Model: A formal specification of the IUT which precisely and unambiguously describes what an implementation may do and not do. The model is usually provided in a formal modelling language which has an IOTS semantics.

Implementation relation: Also called conformance relation, this is the precise mathematical definition of conformance between model and IUT that we are testing for. It defines the concrete decision about applicable inputs for a given state of the model and how that model state has to be updated.

Test generation algorithm: The test generation algorithm depends on the chosen implementation relation and derives test cases from the model by iteratively deciding if an input shall be provided to the IUT and if so, which one, and processing the received outputs from the IUT.

Test case: A test case is a formal description of an experiment to perform on the IUT, whose semantics are again a variation of IOTS with the special states *PASS* or *FAIL*.

Test case execution: The test cases are executed on the actual software, system or device to be tested by translating the abstract transitions to concrete interactions with the IUT.

Test verdict: The test verdict is the concluding result of a concrete execution of a test case (or several test cases), a so-called test run. Corresponding to the final state of the test case this is either "pass" or "fail".

Once all components are in place, the model-based testing process provides means for automated generation and execution of a sound (and, in theory, also complete) set of test cases: no further manual interaction is needed, i.e. no more human errors can be introduced. This means that an implementation which fails a test case is indeed guaranteed to not conform to the given specification and, in theory, for each non-conforming implementation there is a test case detecting that. Test cases are either generated *offline*, i.e. before execution, and then run on the implementation, which enables easy re-execution of test cases, or stepwise *online* during test execution.

IV. EnergyBus

The EnergyBus specification [21] aims at establishing a common basis for the interchange and interoperation of electric devices in the context of energy management systems (EMS). This includes the definition of a family of connectors as well as of appropriate network communication protocols. The central and innovative role of EnergyBus is the transmission and management of electrical power: the purpose of its protocol suite is not just to transmit data, but in particular to manage the safe access to electricity and its distribution inside an EnergyBus network. The EnergyBus protocols are developed by the EnergyBus e.V. association and its members.

A. The Protocols

Conceptually, EnergyBus extends the CANopen architecture with several components, and the EnergyBus protocols are developed in terms of *CANopen application profiles* endorsed by the CiA association [21]. Among these, the "Pedelec Profile 1" (PP1) is very elaborate, targeting a predominant business context, which is also at the centre of ongoing international standardisation efforts as part of IEC/IS/TC69/JPT61851-3. An EnergyBus network consists of several EnergyBus devices, connected by a single CAN bus and sharing a low-voltage auxiliary power line (between 9 and 12 V DC) among so-called *passive devices*. *Active devices* are additionally connected to the main power line ranging from 12 to 250 V DC or from 85 to 265 V AC.

The specification introduces the notion of *virtual devices*, which encapsulate the functionality of a specific, dedicated role in an EMS, e.g. of a battery pack, a motor, or a sensor unit. Among these, the EnergyBus controller (EBC) has a special position. It is responsible for managing the distribution of electric power and for ensuring the electrical safety of the network, especially protection against over- and under-voltage as well as over- and under-current. This is achieved by limiting the power flows according to network parameters and dynamic values communicated by the devices attached. To do so, the EBC sets appropriate limits to other devices and dynamically adjusts them according to the actual settings of the system. Still, every device is ultimately responsible for its own safety and must protect itself from damage by disconnecting from the power lines if necessary. A real (physical) device can combine several, not necessarily different, virtual devices. For example, a public charger can be considered as being composed of a voltage converter, a secondary EBC, and a load monitoring unit, each appearing as one virtual device to the protocol.

B. Certification

An important intention of the EnergyBus approach is to establish a certification process ensuring that customers can freely choose among the set of EnergyBus-compliant devices with guaranteed interoperability, functionality, and safety of their individual set-up. In order to assign such a certificate, a *conformance test plan* [22] is developed, describing how an EnergyBus device is to be tested.

The current version of the conformance test plan is subdivided into three different levels of conformity:

Certificate 1: Device communication test consists of the common CANopen conformance test plan plus an extension for EnergyBus devices, which has been developed by the CiA 454 joint working group. It defines traditional test cases for static checks of a device's communication parameter settings. These test cases are provided as a description of the test purpose, involved parameters, pre- and postconditions, plus pseudocode descriptions of the actual test steps.

Certificate 2: Device hardware test describes hardware properties that need to be checked. These low-level tests are orthogonal to the protocol perspective that we focus on.

Certificate 3: Interoperability test is a manual procedure where the device under test is connected to other EnergyBus-compliant devices in order to understand whether they are interoperable and provide the desired functionality.

Certificate 1 conformance can be checked by combining the CANopen conformance test tool by CiA e.V. (see www.can-cia.org) with the scripting functionality of the CANopen DeviceExplorer tool by emtas (see www.emtas.de).

Certificate 1 will at some point need to be extended by dynamic tests, i.e. longer sequences of interactions will be executed on the device under test, simulating its inclusion and use in an EnergyBus network. Designing, specifying, implementing and running these test cases manually will be time-consuming, cumbersome, and costly. The same holds for checking Certificate 3 conformance. Fortunately, model-based testing can automate checking for these requirements:

V. Model-based Testing for EnergyBus

A formal model of a system's specification is the starting point for model-based testing. Moreover, it is the prerequisite for modern protocol engineering approaches—including the design phase, where it guarantees unambiguous and precise documentation—and enables the use of further formal analysis methodologies like (statistical) model checking.

A. Formal Specification

Since EnergyBus is defined as a layer on top of CANopen, an EnergyBus device is a CANopen device and therefore inherits several CANopen features. Thus EnergyBus documentation as well as the CANopen documentation have to

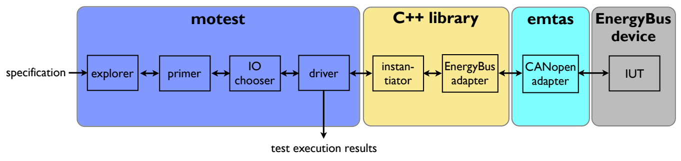

Figure 3. The MOTEST model-based testing setup

be taken into account for formal modelling. Both specifications are provided as informal combinations of text, protocol flow charts, data tables, and finite state automata (FSA). The FSA model the view from other network devices on a particular ENERGYBUS/CANopen device as a graph-based notation for state-dependent interaction capabilities, like IOTS. The basic CANopen specification is defined in [23] (∼150 pages). CAN-open associates to each device several data structures (e.g. the object directory (OD)) and various services for e.g. initial configuration [24], data exchange, and basic communication capability control. On top of this, ENERGYBUS is defined as a collection of 14 documents [21] (about 400 pages, of which 300 only describe OD entries). Most of them give detailed information for particular virtual devices, including specific OD entries and specifying their behaviour as one or several FSA. Several general documents provide the definitions for ENERGYBUS-specific communication and control services.

Our formal model of ENERGYBUS uses the MODEST modelling language. Each FSA is directly translated to a sequence of MODEST process definitions, one for each FSA state, resulting in the corresponding IOTS semantics. The different FSA are intended to run in parallel, loosely coupled. They synchronise over matching pairs of input and output actions and communicate using a combination of handshaking and shared variables. A straightforward modelling of the OD would consist of a two-dimensional array hosting different types of data, which would certainly cause state space explosion in the underlying IOTS. We therefore chose to model only relevant OD entries and implement them depending on the specific nature of each OD entry. *Constants* and *variables* are modelled as plain MODEST constants or variables, respectively, whereby the level of locality in the processes corresponds to the locality of the virtual device combination. *Global variables* serving as communication interfaces are modelled using dedicated MODEST processes with communication actions to update these variables and additional actions that notify all "listeners" of any value update. This approach also avoids the need for explicit models for the different communication services.

Aside from the basic control functionality, the ENERGYBUS protocol is all about data. To overcome the state space explosion problem, we applied several abstraction techniques to appropriate areas of our model. From a conformance testing point of view, such abstractions can be seen as a simple application of action refinement [25], transferring the complexity from the MODEST model to the adapter component.

B. A Modest Testing Tool

Triggered by the needs of ENERGYBUS, model-based testing has recently been added to the MODEST TOOLSET in the form of the MOTEST tool. Its overall structure is depicted in the left part of Figure 3 and follows the approach of [26]. The **explorer** component makes use of the MODEST TOOLSET's infrastructure to access the transition system representation of the given specification. Test generation proceeds according to a particular implementation relation, and this is effectuated by the **primer**. Aside from the classic implementation relation of input-output conformance (ioco), we needed to add an *asynchronous* version that supports bounded message reordering. The decision whether the tester should wait for an output of the implementation, or go ahead in providing an input to it and if so, which one, is taken by the **IO chooser**. The default behaviour provides a simple random number generator with configurable probabilities assigned to the different options, but more enhanced decision taking algorithms are possible. The **driver** steers the test execution algorithm by pipelining received outputs from the IUT to the primer to evolve the model state and request the next action to be taken from the IO chooser, which is then sent to the IUT via the **instantiator**. The driver is the central component where a protocol of the performed test run including the test verdict is compiled. Translating abstract model actions into concrete messages to be sent to the IUT, possibly including data values, and vice versa, is the **instantiator**'s task. The actual communication with the IUT is done via the **adapter**, which implements the required communication protocols. In our particular testing scenario, the adapter is split in two adapters for the different communication layers—ENERGYBUS and CANopen/CAN. The latter is provided by emtas.

We had the opportunity to apply MOTEST to several prototypes and retail devices successfully. It, including the specification models, has been made available free-of-charge to the entirety of the EnergyBus e.V. association and can be used by its members for in-house testing. MOTEST itself is available as part of the MODEST TOOLSET at www.modestchecker.net.

C. Results

We already identified several issues with the (at that time current version of the) ENERGYBUS specification documents in the modelling phase: there were gaps in the specification, preventing some parts of the services to be modelled to a reasonable extent; and there were ambiguities in some parts of the specification, possibly inducing non-interoperability. These have been reported so as to be corrected in standardisation.

The actual test runs then revealed three different types of further errors. The first type were traditional implementation bugs of a non-severe nature that have been reported and fixed. The second type were rooted in two distinct interpretations of the ENERGYBUS basic device initialisation and the core ENERGYBUS device control leading to incompatible implementations. To pinpoint this, we developed two different models of the specification and continued testing with the respective version. The third type of observed errors were intricately related to the hard- and software hierarchy of the test and IUT architecture, i.e. the CAN bus system. They can be viewed as spurious "fail" verdicts rooted in the fact that the different communication layers made the traditional model-based testing assumption of synchronous communication unsound. Asynchronous communication may make responses of the IUT arrive later than expected on the specification side, and the same may happen to stimuli on the IUT side. This is rooted in the latency of communication induced by the ENERGYBUS, CANopen and CAN layers that partially operate concurrently. In addition, we observed that some CAN implementations take the liberty to reorder messages within responses, so that consecutive messages passed by an IUT's application to its local CAN controller may be sent out in reverse order.

The asynchronicity phenomenon triggered the implementation of a new asynchronous model-based testing method: MOTEST now provides an asynchronous version of input-output conformance for sound testing results with all order-preserving CAN implementations. In case of message reordering inside the CAN stack, however, manual inspection would still be needed to definitely rule out spurious "fail" verdicts.

VI. CONCLUSION

In this paper, we have discussed how state-of-the-art formal methods and tools have been applied in the context of the ENERGYBUS specifications for light electric vehicles. We introduced formal modelling as the essential basis for all model-based analyses and elaborated on the characteristics that suitable formalisms have to fulfil. The MODEST modelling language has been introduced as a suitable formalism and its defining features have been presented, together with an overview of the three main model-based analysis methods: simulation, model checking, and model-based testing, which are all available in the MODEST TOOLSET.

Our contributions in the context of ENERGYBUS specification and implementation support the entire process from specification, modelling, verification and certification including both traditional test case programming and model-based testing. Specification inaccuracies as well as programming bugs have been found in tested prototype and retail devices. Based on our insights, documentation and implementations have been improved. The testing process itself motivated us to extend the supported conformance relations in MOTEST to asynchronous testing in order to eliminate spurious errors.

Both the exchange of data values between devices in the network as well as the actual distribution, consumption, and production of electrical energy are important parts of ENERGYBUS. While the MODEST TOOLSET has the needed functionality to formally *model* these characteristics, only simulation and model checking are able to *analyse* models combining them all. So far, our model-based testing approach for the certification process has to be abstract w.r.t. data and the physical quantities and effects involved to be applicable. However, to overcome the state space explosion problem, model-based testing has already been studied with respect to symbolic specifications [27], [28], [29] and we are currently in the process of integrating symbolic testing into MOTEST.

Acknowledgments: This work is supported by the ERC Advanced Grant POWVER (695614), the Sino-German project CAP (GZ 1023), and the 3TU project "Big Software on the Run".

REFERENCES

[1] "Catastrophic Surface Pro 3 battery life finally has its firmware fix," http://arstechnica.com/?p=945575, last access 04/2017.
[2] "Samsung recalls Galaxy Note 7 worldwide due to exploding battery fears," http://theverge.com/2016/9/2/12767670, last access 04/2017.
[3] "Basis Peak watches recalled," http://techcrunch.com/2016/08/03/basis-peak-watches-recalled-due-to-overheating/, last access 04/2017.
[4] "Important: Medical device correction, EnRhythm pacemakers," http://www.medtronic.com/enrhythm-advisory/downloads/enrhythm-battery-issues_physician-letter.pdf, last access 04/2017.
[5] "Qualitätsprobleme bei E-Bikes: Schlappe Akkus, anfällige Elektronik," http://www.spiegel.de/auto/aktuell/a-790142.html, last access 04/2017.
[6] E. M. Hahn, A. Hartmanns, H. Hermanns, and J. Katoen, "A compositional modelling and analysis framework for stochastic hybrid systems," *Formal Methods in System Design*, vol. 43, no. 2, pp. 191–232, 2013.
[7] A. Hartmanns and H. Hermanns, "The Modest Toolset: An integrated environment for quantitative modelling and verification," in *TACAS*, ser. LNCS, vol. 8413. Springer, 2014, pp. 593–598.
[8] A. Hartmanns and H. Hermanns, "In the quantitative automata zoo," *Sci. Comput. Program.*, vol. 112, pp. 3–23, 2015.
[9] P. R. D'Argenio and J.-P. Katoen, "A theory of stochastic systems part I: Stochastic automata," *Inf. Comput.*, vol. 203, no. 1, pp. 1–38, 2005.
[10] R. Alur and D. L. Dill, "A theory of timed automata," *Theor. Comput. Sci.*, vol. 126, no. 2, pp. 183–235, 1994.
[11] T. A. Henzinger, "The theory of hybrid automata," in *LICS*. IEEE Computer Society, 1996, pp. 278–292.
[12] M. Sijtema, A. Belinfante, M. Stoelinga, and L. Marinelli, "Experiences with formal engineering: Model-based specification, implementation and testing of a software bus at Neopost," *Sci. Comput. Program.*, vol. 80, pp. 188–209, 2014.
[13] C. E. Budde, C. Dehnert, E. M. Hahn, A. Hartmanns, S. Junges, and A. Turrini, "JANI: Quantitative model and tool interaction," in *TACAS*, ser. LNCS, vol. 10206. Springer, 2017, pp. 151–168.
[14] A. Hartmanns, H. Hermanns, and M. Bungert, "Flexible support for time and costs in scenario-aware dataflow," in *EMSOFT*. ACM, 2016.
[15] F. van den Berg, J. Hooman, A. Hartmanns, B. R. Haverkort, and A. Remke, "Computing response time distributions using iterative probabilistic model checking," in *EPEW*, ser. LNCS, vol. 9272. Springer, 2015.
[16] H. L. S. Younes and R. G. Simmons, "Probabilistic verification of discrete event systems using acceptance sampling," in *CAV*, ser. LNCS, vol. 2404. Springer, 2002, pp. 223–235.
[17] T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet, "Approximate probabilistic model checking," in *VMCAI*, ser. LNCS, vol. 2937. Springer, 2004, pp. 73–84.
[18] A. Hartmanns, H. Hermanns, and P. Berrang, "A comparative analysis of decentralized power grid stabilization strategies," in *Winter Simulation Conference*, 2012.
[19] H. B. Graf, H. Hermanns, J. Kulshrestha, J. Peter, A. Vahldiek, and A. Vasudevan, "A verified wireless safety critical hard real-time design," in *WOWMOM*. IEEE Computer Society, 2011.
[20] J. Tretmans, "Model-based Testing with Labelled Transition Systems," in *Formal Methods and Testing*, R. M. Hierons, J. P. Bowen, and M. Harman, Eds. Springer-Verlag, 2008, pp. 1–38.
[21] CAN in Automation Int. Users and Manufacturers Group e.V. and EnergyBus e.V., *CiA 454 Draft Standard Proposal Application profile for energy management systems – doc. series 1-14, v. 2.0.0*, 2014.
[22] CAN in Automation Int. Users and Manufacturers Group and EnergyBus e.V., *CiA 454 – Conf. Test Plan Pedelec Profile 1, v. 0.11*, 2017.
[23] CAN in Automation Int. Users and Manufacturers Group e.V., *CiA 301 CANopen Application Layer and Comm. Profile, v. 4.2.0*, 2011.
[24] CAN in Automation Int. Users and Manufacturers Group e.V., *CiA 305 Layer setting services (LSS) and protocols, v. 3.0.0*, May 2013.
[25] M. Bijl, A. Rensink, and J. Tretmans, "Action Refinement in Conformance Testing," in *Testing of Communicating Systems*, ser. LNCS. Springer, 2005, vol. 3502, pp. 81–96.
[26] J. Tretmans and E. Brinksma, "TorX: Automated Model Based Testing – Côte de Resyste," 2003.
[27] L. Frantzen, J. Tretmans, and T. Willemse, "Test gen. based on symbolic specifications," in *FATES*, ser. LNCS, vol. 3395. Springer, 2004.
[28] L. Frantzen, J. Tretmans, and T. Willemse, "A symbolic framework for model-based testing," in *FATES and RV*, ser. LNCS, vol. 4262. Springer, 2006, pp. 40–54.
[29] S. Chakrabarti and S. Ramesh, "Symtest: A framework for symbolic testing of embedded software," in *India Softw. Eng. Conf.* ACM, 2016.