designs

Article

Model Testing of Complex Embedded Systems Using EAST-ADL and Energy-Aware Mutations

Eduard Paul Enoiu * and Cristina Seceleanu *

Mälardalen University, Department of Networked Embedded Systems, 722 20 Västerås, Sweden
* Correspondence: eduard.paul.enoiu@mdh.se (E.P.E.); cristina.seceleanu@mdh.se (C.S.)

Received: 18 November 2019; Accepted: 13 February 2020; Published: 19 February 2020

Abstract: Nowadays, embedded systems are increasingly complex, meaning that traditional testing methods are costly to use and infeasible to directly apply due to the complex interactions between hardware and software. Modern embedded systems are also demanded to function based on low-energy computing. Hence, testing the energy usage is increasingly important. Artifacts produced during the development of embedded systems, such as architectural descriptions, are beneficial abstractions of the system's complex structure and behavior. Electronic Architecture and Software Tools Architecture Description Language (EAST-ADL) is one such example of a domain-specific architectural language targeting the automotive industry. In this paper, we propose a method for testing design models using EAST-ADL architecture mutations. We show how fault-based testing can be used to generate, execute and select tests using energy-aware mutants—syntactic changes in the architectural description, used to mimic naturally occurring energy faults. Our goal is to improve testing of complex embedded systems by moving the testing bulk from the actual systems to models of their behaviors and non-functional requirements. We combine statistical model-checking, increasingly used in quality assurance of embedded systems, with EAST-ADL architectural models and mutation testing to drive the search for faults. We show the results of applying this method on an industrial-sized system developed by Volvo GTT. The results indicate that model testing of EAST-ADL architectural models can reduce testing complexity by bringing early and cost-effective automation.

Keywords: model testing; mutation testing; energy consumption; EAST-ADL

1. Introduction

Embedded products are widely used in many industries. For example, embedded systems are used in automotive companies in the implementation of vehicle functions (e.g., ABS, electronic stability) [1]. Such functions contribute to the complexity of developing the entire vehicle system [2], making the verification and validation of new functions more problematic due to the interconnections between both functional and non-functional requirements posed on the whole system. For instance, a structural or behavioral update in the software or the replacement of a software or hardware part can influence the consumption of resources [1]. In this case, just showing the overall system's functional correctness is not enough. One would need to verify that the system meets its non-functional (also known as extra-functional) requirements, such as energy consumption, memory allocation and real-time perfomance. In addition, although testing is arguably the most used verification and validation technique, for these complex systems testing is highly expensive when performed on the actual system. Given the increasing demand in embedded systems for low-energy computing [3], early testing the energy consumption becomes an increasingly important issue. To ensure the quality of service of embedded systems and to estimate its performance early in the development process, testing the behavior of the system with respect to its supplied energy budget as well as testing for the worst-case energy consumption is very important.

In this paper we outline a method that targets these challenges by bringing extra-functional testing of these complex embedded systems by moving the creation, execution of test cases earlier in the development process at an abstract architectural modelling level. Architectural models are used to represent relevant aspects of system behavior, environment, structure, and properties and are used as a basis for test generation, performance, and analysis. Briand et al. [4,5] refers to such techniques as model testing, since this kind of approaches aim to identify faults by executing test cases on models and sampling the input space. This in contrast to other verification and validation techniques such as model checking that attempt to exhaustively explore the model state space and check the correctness of the models against some given properties. Our goal is to tackle the challenges of testing such complex systems by developing an approach that provides, early-on in the development process, confidence about the system resource consumption by identifying and executing a selected set of test cases, from the whole test execution space, where faults are more likely to lie. In the automotive domain, modelling the architectural aspects including the resource consumption of complex embedded systems at high levels of abstraction is necessary. Architectural description languages such as EAST-ADL (EAST-ADL stands for Electronic Architecture and Software Tools-Architecture Description Language.) [6] are used to represent both hardware and software functions and extra-functional information (e.g., timing properties and resource consumption). If we assume energy as the resource of interest, the annotations of energy consumption in EAST-ADL models can be used to create test cases for feasibility and worst-case energy consumption that can be useful in the detection of faults.

This article is an extended version of a conference paper [7] in which we have demonstrated how architectures described in the EAST-ADL language can integrated into a testing approach in order to evaluate energy properties. Based on this previous work, in this study we present a novel mutation-based approach for model testing and evaluate its efficiency and effectiveness on an industrial use case. In particular, by selecting test cases using mutation testing [8], we propose a method for automatically generating and selecting test cases based on the concept of energy-aware mutants–small architectural model syntactic modifications designed to mimic real energy faults. Test cases where a certain behavior can be distinguished from its mutations are sensitive to changes in the architectural model and are therefore considered good at detecting faults.

We apply this method on an embedded system modeled in EAST-ADL after transforming it into a network of priced timed automata [9]. In particular, we select test suites based on random model executions that show the energy cost using UPPAAL SMC [10], the statistical extension of the UPPAAL model checker. We show how to seed faults in the EAST-ADL model and evaluate each generated test suite's energy-related fault detection capability. To illustrate the efficiency and effectiveness of our test generation method, we carry out an evaluation, using an industrial system modeled in EAST-ADL architectural language. The results of this study suggest that model testing is efficient in terms of test generation time and number of generated and selected test cases.

To summarize, the main contributions of this paper are:

- The identification of energy-aware mutation operators for mutation testing of EAST-ADL models.
- An approach for mutation test generation of EAST-ADL models using a statistical model checker.
- An evaluation of the method on a Brake-by-Wire industrial system.

The rest of the paper is organized in the following sections. In Section 2 we overview the preliminaries needed to comprehend our contribution, including architecture-based testing and mutation testing, the EAST-ADL architectural language, UPPAAL SMC and priced timed automata. The main contribution of the paper is our method for automatically generating energy-aware test cases using EAST-ADL models described in Section 3, and its application on the Brake-by-Wire system as well as the experimental results presented in Section 4. We conclude the paper and present the future work in Section 7.

2. Background

Major aspects of architecture-based and mutation testing, the language of EAST-ADL and UPPAAL SMC will be discussed in this section. These aspects are put in the scope of the contributions of this study.

2.1. Architecture and System-Level Testing

Architectural models are created during system development using components and connectors representing the whole system and its high-level structure [11]. The aim of testing using architectural designs as input is to verify whether the system meets its design specifications. This type of testing is also known as system-level testing [12] and its main purpose is to discover early-on architectural design problems, but also the overall system behaviour. Testing, at this level, aims to address such test goals as overall functionality, real-time properties, robustness and performance [11]. Testing extra-functional properties (e.g., bandwidth, energy and memory) [13] are crucial aspects during development and need to be addressed continuously when testing. Specifically, we focus on testing for energy consumption based on architectural models. Due to the intertwining of software and hardware and the complex interactions with the external environment, it is challenging to apply conventional testing methods directly to the real embedded systems. When testing for extra-functional properties at the software architecture level, models are annotated with energy consumption properties.

The aim of testing at architectural level based on the energy consumption is to find faults in the performance of an actually developed system in terms of its subsystems and interactions before the actual code implementation. The use of such architectural models for testing enables the execution of a large number of test cases and increases the chances of uncovering faults.

2.2. Mutation Testing

Mutation testing is the technique used for creating a faulty implementations (usually in an automated way) to examine a test-suite's ability to detect faults [8]. A mutant is a new version of a program created by making in the original program a small change. For instance, a mutant is created in a program by replacing an operator with another, negating a variable, or changing a constant's value. The execution of a test suite on the resulting mutant will produce a different output than the original program, in which case the test suite kills the mutant. In order to measure the mutant detection capability of the written test suite, a mutation score is calculated using the automatically seeding all mutants and executing the test cases on each mutant. One can compute a mutation score based on an output-only oracle (i.e., expected outputs) against all the generated mutants by calculating the ratio of mutants killed to the total number of mutants. Just et al. [14] showed that if a test suite can detect or kill most mutants, it can also detect real software faults, thus providing evidence that the mutation score is a fairly good proxy for real fault detection ability. Mutation testing has been widely used at lower levels of testing and mostly on implementation models. Even if there are some studies that have applied this technique on specification models [15–18] for designing behavioral faults, there is a lack of methods that target architectural models and extra-functional aspects for model mutation testing. No attempt has been made to propose and evaluate mutation testing for EAST-ADL models. This motivated us to develop an automated approach to test generation and model testing using mutations aimed at this kind of architectural model.

2.3. EAST-ADL Architectural Language

EAST-ADL [6] is an AUTOSAR-compatible (AUTOSAR is a standard for AUTomotive Open System ARchitecture and was developed by several manufacturing companies.) architectural description language intended to be used in the development of automotive embedded systems. A system can be described at four levels of abstraction, as follows: (i) the Vehicle Level describes the external features at the highest level of abstraction, (ii) the Analysis Level describes the abstract functionality of the system, (iii) the Design Level describes more details in the functional representation of the architecture and the hardware allocation of these onto the platform, and (iv) the Implementation Level provides the AUTOSAR-compliant code.

At each level of abstraction, the model is composed using components (i.e., FunctionType) which describe the functionality of the system. Each FunctionType contains: (i) Ports that receive and provide data, (ii) a trigger (i.e., time-based or event-based), and (iii) an internal behavior. Each of these components is instantiated as a FunctionPrototype. The execution of each FunctionPrototype uses the "read-execute-write" semantics, and the internal behavior is defined using different languages (e.g., Simulink, UML, UPPAAL PORT timed automata [19,20]). In this study we use the models at the design level, containing the Functional Design Architecture (FDA) and Hardware Design Architecture (HDA) annotated with non-functional properties. The design model can be annotated with a GenericConstraint property representing the energy utilization.

2.4. UPPAAL SMC and Priced Timed Automata

UPPAAL SMC [10] is an extension of UPPAAL, that supports the analysis of non-functional properties for networks of priced timed automata with stochastic semantics. Statistical model-checking is used to generate stochastic simulations and estimate probabilities and probability distributions over time with a certain level of confidence, so the analysis scales better than symbolic model-checking in verification of realistic industrial models. Specifically, statistical model checking samples executions using statistical inference methods to decide whether the model executions satisfy a property given a certain confidence. In this paper we use statistical model checking and UPPAAL SMC in a black-box manner for execution of models as well as producing probabilistic estimates about the correctness of the generated models.

Priced timed automata (PTA) are used in UPPAAL SMC and are extensions of timed automata with cost variables that can evolve at integer rates (also $\neq 1$). These are used for representing the energy consumption. The energy usage is modeled using a function $P : (L \cup E) \to \mathbb{N}$, where L is a finite set of locations, and E is the set of edges, which assigns costs to both locations and edges. A network of PTA (NPTA) is described as a composition of n PTA over clocks and actions; the PTA use send–receive actions (i.e., send $b!$ is complementary to receive $b?$) and shared variables are used in guards.

Let X be a finite set of clocks and $B(X)$ the set of guards, which are finite conjunctions of atomic guards of the form $x \bowtie n$, where $x \in X$, $n \in \mathbb{N}$, and $\bowtie \in \{<, \leq, =, \geq, >\}$. A (Linear) PTA over clocks X and actions Act is a tuple $(L, l_0, X, V, I, Act, E, P)$ where L is a finite set of locations, l_0 is the initial location, X is set of clocks, V is a set of data variables, $I : L \to B(X)$ assigns invariants to locations, Act is a set of actions, $E \subseteq L \times B(X, V) \times Act \times R \times L$ is the set of edges (where R denotes the reset set, i.e., assignments to manipulate clock- and data variables), and $P : (L \cup E) \to \mathbb{N}$ assigns costs to both locations and edges. In the case of $(l, g, a, r, l') \in E$, we write $l \xrightarrow{g,a,r} l'$.

The semantics of PTA is defined as a transition system over states (l, u), with the initial state (l_0, u_0), where u_0 assigns all clocks in X to zero. There are two kinds of transitions:

(i) Delay transitions: $(l, u) \xrightarrow{d,p} (l, u \oplus d)$, where $u \oplus d$ is the result obtained by incrementing all clocks of the automata with the delay amount d, and $p = P(l) * d$ is the cost of performing the delay, and

(ii) discrete transitions: $(l, u) \xrightarrow{d,p} (l', u')$, corresponding to taking an edge $l \xrightarrow{g,a,r} l'$ for which the guard g is satisfied by u. The clock valuation u' of the target state is obtained by modifying u according to updated r. The cost $p = P(l, g, a, r, l')$ is the priced associated with the edge.

A network of PTA $A_1 \parallel ... \parallel A_n$ is expressed as a composition of n PTA over X and Act, using synchronization actions and shared variables that can be used in guards and transitions. UPPAAL SMC uses an extended Weighted Metric Temporal Logic (WMTL) [21] for performing hypothesis testing, which checks if the probability to reach a state ϕ within cost $x \leq C$ is greater or equal to a certain threshold $p : Pr(\Diamond_{x \leq C}\phi) \geq p$.

A trace σ of a PTA is a sequence of delays, actions, and transitions:
$\sigma = (l_0, u_0) \xrightarrow{a_1, p_1} (l_1, u_1) \xrightarrow{a_2, p_2} ... \xrightarrow{a_n, p_n} (l_n, u_n)$, where the cost of performing σ is $\Sigma_{i=1}^{n} p_i$.

3. A Model Testing Method for Energy-Aware Testing Using EAST-ADL

In this section we introduce our model testing method which uses energy consumption objectives to select test suites using a statistical model checker based on the created simulations. The framework is based on the transformation of the EAST-ADL model into a network of priced timed automata (PTA) [22]. It is composed of several steps, mirrored in Figure 1:

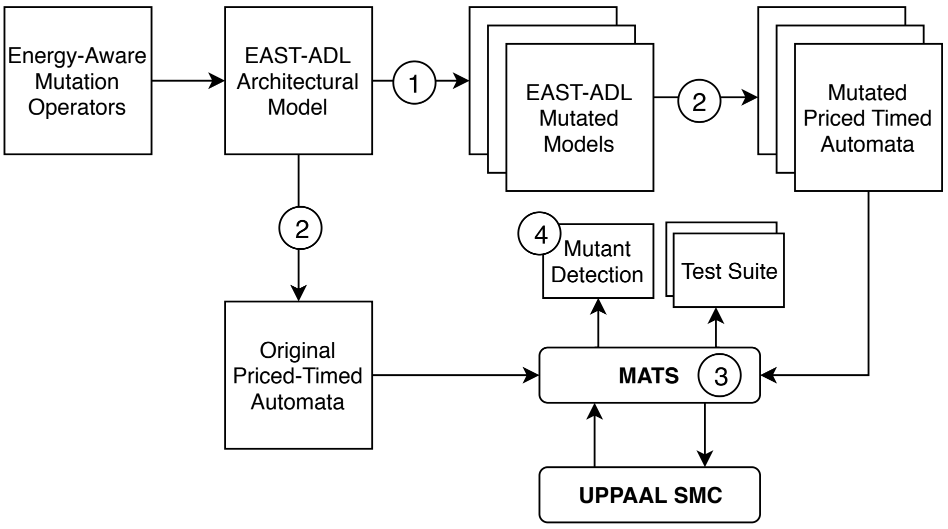

Energy-Aware Mutation Operators
EAST-ADL Architectural Model
1
EAST-ADL Mutated Models
2
Mutated Priced Timed Automata
2
4
Mutant Detection
Test Suite
Original Priced-Timed Automata
MATS
3
UPPAAL SMC

Figure 1. An overview of the test suite generation and evaluation method for energy consumption based on Electronic Architecture and Software Tools Architecture Description Language (EAST-ADL) architectural models and mutation testing.

1. Mutant Generation. This first step (described in detail in Section 3.1) is used for generating small syntactic changes (mutants) to the architectural description based on a set of mutation operators (e.g., mimicking architectural energy errors). The output of this step is a set of new versions of the original EAST-ADL model, each one containing an inserted change. When implementing this step, a set of operators needs to be available based on the desired types of mutants. Different set of operators were proposed in the scientific literature for both models and code [18,23] for mutation testing using the UPPAAL model checker. We show how mutant generation is implemented for the EAST-ADL language as an input for MATS and UPPAAL SMC.
2. EAST-ADL to PTA Transformation. The second step (described in detail in Section 3.2) is used for transforming the EAST-ADL model to PTA. The output of this step are PTA models containing the original structure and behavior of the EAST-ADL together with all inserted mutants and annotated with energy consumption information to be used by UPPAAL SMC for test-case generation and selection.
3. Test Suite Generation. The third step (described in Section 3.3) uses the MATS tool to generate a set of test cases by using the UPPAAL SMC ability to generate simulations. We show how a test simulation is obtained using a property expressed as a UPPAAL SMC simulation property.
4. Mutant Detection. The fourth step (described in Section 3.4) involves the instrumentation of the model with detection instructions for each mutant. This means that the monitor for mutation detection is used to record the execution and detection of each mutant.

We discuss these steps in further detail in the following sections by applying this approach to running examples.

Overall, we use an EAST-ADL system architectural model and mutation testing to automatically generate test suites for model testing the energy consumption. Our test generation method aims to use mutations in energy consumption in EAST-ADL to select test suites automatically using various random simulations.

3.1. Energy-Aware Mutant Generation

In this paper we assume a resource r for an EAST-ADL component represents the accumulated resource usage up to some point in time. By using this assumption, resources of this kind are categorized as discrete or continuous [24] . Energy in EAST-ADL is a continuous resource that can evolve linearly in time ($energy(t) = n \times t$, where $n \in \mathbb{N}$ and t is the time elapsed during system execution). In EAST-ADL components the resource usage is defined as the total energy consumption of the system as $energy_{total}(t) = \sum_{i=1}^{m} energy_i(t)$, where m is the number of components. Based on a predefined set of mutation operators, faults are injected. These mutants should represent naturally-occurring faults influencing the energy consumption. The general principle underlying mutation analysis is that the faults generated using the operators described in Table 1 represent the mistakes that architects often make while modelling in EAST-ADL that directly influence the energy consumption.

We propose a set of mutation operators and they are applied on all the EAST-ADL model elements that could influence the energy consumption in the following three categories:

- EAST-ADL resource annotation (i.e., Energy Replacement Operator (ero)). In this case, we insert a fault in the generic constraint of a function prototype by changing the original annotation. These types of mutations intend to model the errors in the energy consumed by each component.
- Timing Behavior of an EAST-ADL component (i.e., Period Replacement Operator (pro), Execution Time Replacement Operator (etro)) can be modified by changing the period and execution time constraints. The period and execution time value stand as integer values in the constraint.
- Functional Architecture Structure (i.e., Component Removal Operator (cro), Component Insertion Operator (cio), and Triggering Pattern Replacement Operator (tro)). We change the architectural elements in EAST-ADL by removing or inserting components that influence the energy consumption as well as modifying the triggering of each component.

In the case of the CRO mutation operator, a component is directly removed when we encounter an entry, computation and exit function prototype. An entry function prototype in an EAST-ADL model is a component that has at least one port receiving inputs externally. The computation function prototype has all input and output ports connected with other function prototypes in the same level of system abstraction. An exit function prototype has at least one output port sending data flows out of the actual system. When a component is removed, the connections must also be removed so the system remains well formed (compilable).

Table 1. Description of each mutation operator and the elements in EAST-ADL that are modified.

Mutation Operator	**Description**	**EAST-ADL Element**
Energy Replacement Operator (ero)	The operator is applied where an energy value occurs occurs, i.e., as an annotation of each component. The operator is applied by replacing a value of an energy constraint connected to a component (e.g., replacing a value ($value = 3$) with its boundary values (e.g., $value = 2$)).	GenericConstraint
Period Replacement Operator (pro)	The operator applies the mutations in each period constraint value. The operator is only applied in the components triggered periodically. The operator is applied by replacing a value of the period constraint connected to a component (e.g., replacing a value ($value = 20$) with its boundary values (e.g., $value = 19$)).	PeriodConstraint
Execution-Time Replacement Operator (etro)	The operator applies the mutations in each execution time constraint value. The operator is applied by replacing a value of the execution time constraint connected to a component (e.g., replacing a value ($value = 3$) with its boundary values (e.g., $value = 4$)).	ExecTimeConstraint
Component Removal Operator (cro)	This operator models errors related with missing components. This operator removes each component together with its constraints and connects the inputs of this component to the next component in the system.	FunctionPrototype
Component Insertion Operator (cio)	This operator models errors related with duplicated components. This operator adds a duplicated component together with its constraints and connects this component in the same configuration as the original one.	FunctionPrototype
Triggering Replacement Operator (tro)	The operator applies the mutations in each component triggering pattern. The operator is applied by replacing the periodic pattern with a event pattern connected to a component and vice versa.	FunctionPrototype, PeriodConstraint

In Figure 2 we show examples of mutations for each mutation operator in Table 1. For each category of mutations, a mutant operator was chosen to provide a small-scale example of the application to a EAST-ADL model. For example, in Figure 2d for the CRO operator changing the structure of the model within a system is a likely operation within an EAST-ADL project. Depending on how the mutation operator is used, the other inports and connections have to be updated. In this case, CRO removes FP2 and the connection between FP1.Port2 and FP3.Port1. To avoid compilation problems a component is removed together with its control and connection structures.

These mutation operators are systematically applied to the entire EAST-ADL model (i.e., components, ports, connections) each simulating one syntactic change resulting in a set of energy-aware mutants. During the execution of a test, energy consumption can be measured by the use of a statistical model checker, and represented as a consumption of a continuous resources where the rate of consumption over time is constant. A temporal sequence of energy values can have different shapes, depending on the sampling rate of the measurement and the energy consumption behavior.

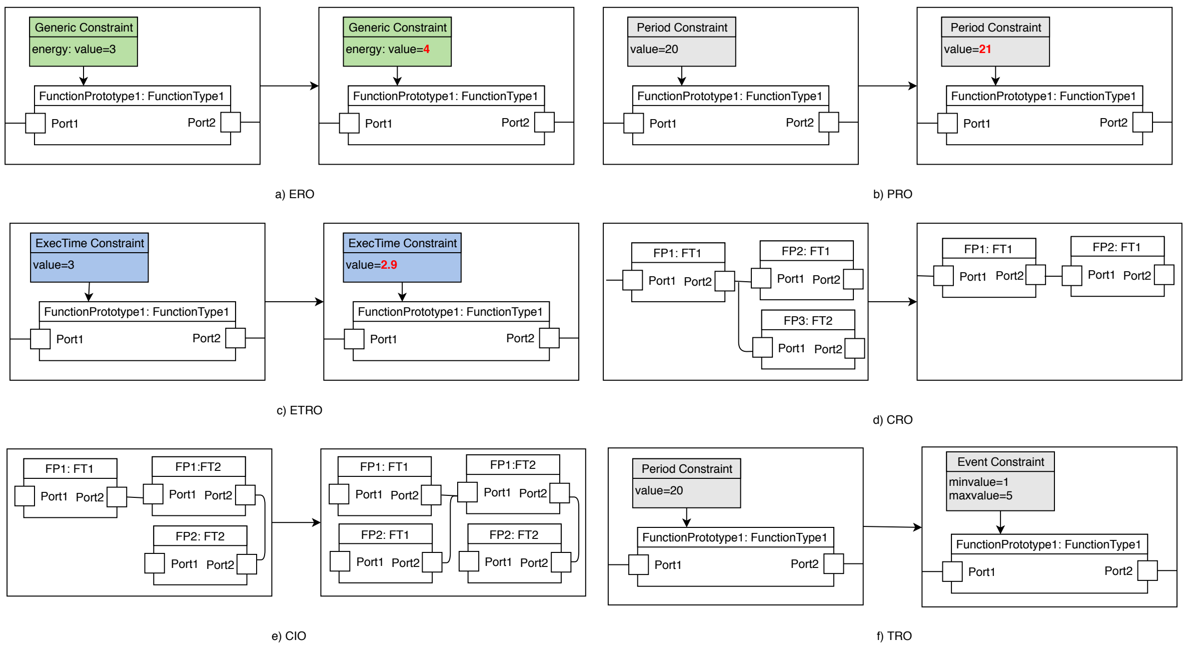
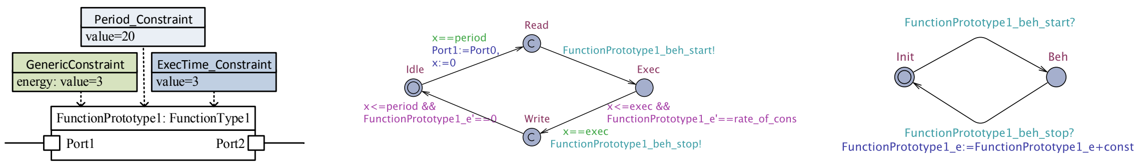


Figure 2. Examples of mutations for each mutation operator for EAST-ADL architectural models.

3.2. EAST-ADL to Priced-Timed Automata Transformation

We transform an EAST-ADL model (with annotations of energy consumption) into a PTA model. We use a small-case example in Figure 3 to show the transformation for a generic EAST-ADL FunctionPrototype. Every component in EAST-ADL is automatically converted into a network of two PTAs: An interface automaton representing the component interface, and a behavior automaton representing the internal behavior. The PTA interface contains the triggering of each component, timing information and energy annotations.


Period_Constraint
value=20
GenericConstraint
energy: value=3
ExecTime_Constraint
value=3
FunctionPrototype1: FunctionType1
Port1
Port2
Read
x==period
Port1:=Port0,
x:=0
Idle
FunctionPrototype1_beh_start!
Exec
x<=period &&
FunctionPrototype1_e'==0
Write
x<=exec &&
FunctionPrototype1_e'==rate_of_cons
x==exec
FunctionPrototype1_beh_stop!
FunctionPrototype1_beh_start?
Init
Beh
FunctionPrototype1_beh_stop?
FunctionPrototype1_e:=FunctionPrototype1_e+const

(**a**) A generic FunctionPrototype. (**b**) The interface timed automaton. (**c**) Behavior template.

Figure 3. An example of a generic interface timed automaton and a behavior template for an EAST-ADL component.

In practice, each FunctionPrototype is translated into a network of two syncronized automata (as shown in Figure 3): An interface automaton containing the ports of each EAST-ADL component and a behavior automaton representing the internal discrete and continuous behavior. Each *FunctionPrototype* is defined as an automaton with four locations: (i) *Idle*, (ii) a *Read* location used for updating the internal variables according to the values on the input ports, (iii) a *Exec* location used for triggering the internal *Behavior*, and (iv) a *Write* location allowing the update of output ports based on the internal variables values. Each interface is triggered based on the triggering annotation *Trigg* associated with each EAST-ADL *FunctionPrototype*. The energy starts to be consumed when information from the input ports is read until the component writes the information to the output ports. The energy consumed by each component increases with time during execution and is modeled as a cost "c" in the PTA ($c(t) = n_c \times t$, where $n_c \in \mathbb{N}$ is the rate of consumption over time t). When the component is idle ($c'(t) = 0$) energy is not consumed. In order to calculate the overall consumed energy we use an automaton to compute the energy used by the system based on the energy consumed by each component.

The GenericConstraint is used in the EAST-ADL model for annotating the energy consumption. By using *energy* in the GenericConstraint annotation for the behavior of the PTA model, we can measure the energy consumption inside each component. Specifically, we use a monitor automaton that includes all of the EAST-ADL model's energy annotations. This monitor is a PTA that supervises the system execution by using two synchronization channels: FunctionPrototype_beh_start and FunctionPrototype_beh_stop.

3.3. Test Suite Generation

To create executable test cases, we use traces obtained during simulation. At each predefined time unit, test values are obtained by extracting the input parameters and energy values from the these simulations. The generation of test suites is essentially the creation of signals using the generated simulations. Since the input signal search space is very large, we randomly select input signals that change over a certain predefined period of time using ordered signal sequences.

Extra-functional aspects in EAST-ADL, such as the energy consumption, are often more difficult to generate test than for functional properties. Embedded systems often require large amounts of energy to be consumed. Irregular and heavy use of energy could result in inadequate functionality of the system that keeps essential components running. The estimation of the allocated energy budget can be calculated using the energy consumption annotated in EAST-ADL. The system will complete its execution if the actual energy consumption of the system does not exceed its energy budget. Otherwise, the budget has been exhausted during the execution of the system. As a result, we concentrate on test queries that simulate the nominal but also the worst-case energy consumption for a EAST-ADL model.

While UPPAAL SMC's is used to provide statistical guarantees based on a series of system simulations, it is also suitable to use the input parameters and the consumed energy as values in test cases during individual system simulations. The simulation depends on the number of runs (n) and the upper time limit for the number of runs (*bound*). By extracting the input parameters and the energy values from these simulation traces, we create executable test cases using the MATS tool [25]. In practice, we use UPPAAL SMC's ability to generate simulation traces, which we transform into executable test cases using the MATS tool [25], by extracting the parameters and the energy values at predefined time points. Each test input is a vector of signals where the model's time-dependent behavior is executed using an ordered signal sequence. UPPAAL SMC is used by MATS to obtain traces of simulations over a predefined number of system model runs. A simulation can be formulated in UPPAAL SMC as the property:

$$simulate\ n[bound]\{E_1, .., E_k\},$$

where n is the number of simulations to be performed, *bound* is the time bound on the simulations, and $E_1, .., E_k$ are the monitored expressions.

We execute the generated test cases on each mutant and collect the simulation traces containing the energy values. On both the original model and its mutated versions, each test case is executed. We exclude test cases that do not contribute to the mutation score in order to minimize the final set of test cases [25]. The generated simulation traces are transformed into executable test cases sampling the simulation trace (as shown in the small-scale example in Figure 4a). Based on the generated data points we use intermediate values at predefined sample points and split the simulation trace in two: A set of sampled inputs used to trigger the system under test and a set of sampled expected energy consumption output.

In addition, during this phase we can generate test case with the worst-case energy consumption. Using UPPAAL SMC for statistical analysis we can obtain the peak energy value which eventually reaches a certain behavior in time. This problem is reduced to trying to maximize the energy cost function to satisfy the following property:

$$E[bound; n](max : energy)$$

where *bound* is the time bound of these simulation traces, n is the number of runs, and *energy* represents energy cost. The worst-case energy consumption analysis calculates the simulation cost for reaching a predefined system behavior. At this stage, feasibility analysis is used to verify whether the energy consumption is still within the maximum energy value provided by the worst-case analysis. The verification is accomplished by evaluating the energy distribution as a probability evaluation, as follows:

$$Pr[bound](\psi)$$

where *bound* is the time bound in all the simulation runs, and ψ is a property in the form "Eventually p", where p is a state predicate. We can select the required test suites based on this analysis. Since the tester may have limited time in practice to test all scenarios, one can choose from the randomly generated test cases the significant and potentially problematic simulations.

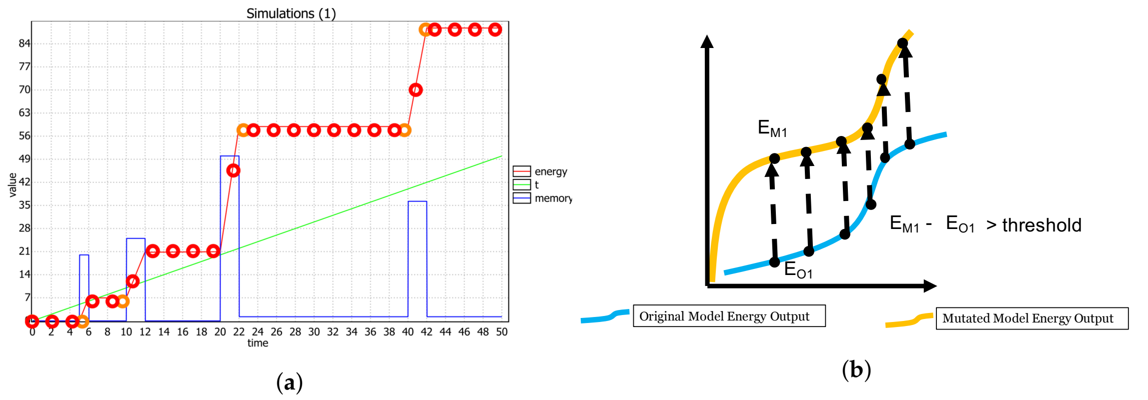


Figure 4. Test Generation and mutant detection for energy consumption. (**a**) An example of a sampled simulation trace obtained from UPPAAL SMC. (**b**) An example of applying mutation detection on two traces based on a predefined threshold.

3.4. Energy-Aware Mutant Detection

A fault is considered to be detected by a test suite if at certain time points the energy values differ drastically. In this way we increase the likelihood of evaluating a certain energy-related behavior (e.g., if the energy differs significantly from the expected result). We assume in this study that small deviations from the specified energy values can be acceptable, test engineers are likely to identify a fault if the energy deviations are substantial.

From the generated test cases, inputs and output values are extracted and used for mutation detection. The sequence of inputs in each test case is automatically inserted in a set of generated mutated models. The mutated models are simulated with the extracted inputs to obtain sets of outputs. The actual outputs extracted from these mutated models for each test case are compared to the expected outputs in order to determine the test case ability to kill (detect) any difference between the mutated models and the original one.

To exemplify this step, we show an example of a test suite that detects a mutant if the energy signal varies significantly from the expected energy values at certain time points (as shown in Figure 4b). In practice, we use a quantitative measure of mutant detection to measure the mutant-revealing capability of a test suite. Let a test case TC be created for a mutated system model M, and let $E_M = E_{M1}, ..., E_{MN}$ be the set of energy signals generated by simulating M for the inputs in TC and sampled at N time points. Let $E_O = E_{O1}, ..., E_{ON}$ be the corresponding expected energy signals. We use a threshold to verify if the distance is greater than this threshold between each value of EO and EM at each time point. If there is at least one energy value in EM for which the distance is greater than the expected threshold, we say that we can detect the mutant M. Otherwise, the injected fault is not detected by TC.

4. An Experimental Evaluation on the Brake-By-Wire System

We experimentally evaluate this model testing method by applying on an industrial system provided by Volvo Group Trucks Technology in Sweden. We perform experiments on a Brake-by-Wire (BBW) industrial system and evaluate the applicability of model testing in creating test cases based on energy-aware mutation testing. Additionally, by using automatically seeded faults, we examine the energy-related detection capability of the generated test suites. We start by injecting a set of faults into the original model to facilitate the assessment of fault detection. For the creation of faults, we rely on energy consumption mutation operators shown in Table 1.

Specifically, we use a PTA model, run the created test suites and collect the traces from simulations containing the energy values for each faulty model. Each test suite is executed on both the original model and its faulty counterpart to calculate the fault detection score. A fault is considered to be detected when energy results differ between executions. As an additional step, we can use the analysis result by removing the test cases that do not contribute to the mutation score of the entire test suite.

4.1. Case Description

The work proposed in this evaluation targets the Design Level in EAST-ADL (i.e., Functional Design Architecture (FDA) and Hardware Design Architecture (HDA) system aspects). The model can be extended with a GenericConstraint annotation, which allows the architect to model the energy consumption. Figure 5 presents a part of the BBW system at Design Level which is allocated to a pedal ECU. This model is extended with annotations for energy consumption as a GenericConstraint. The BBW system is a braking system that contains an anti-lock braking (ABS) feature and no mechanical connections between the brake pedal and the brake actuators. A brake pedal-mounted sensor reads its position, which is used to calculate the desired global brake torque. At each wheel, sensor values are used to calculate the wheel speed used by the ABS algorithm along with the brake torque and the approximate speed of the vehicle to determine the real brake torque to be sent to the actuators.

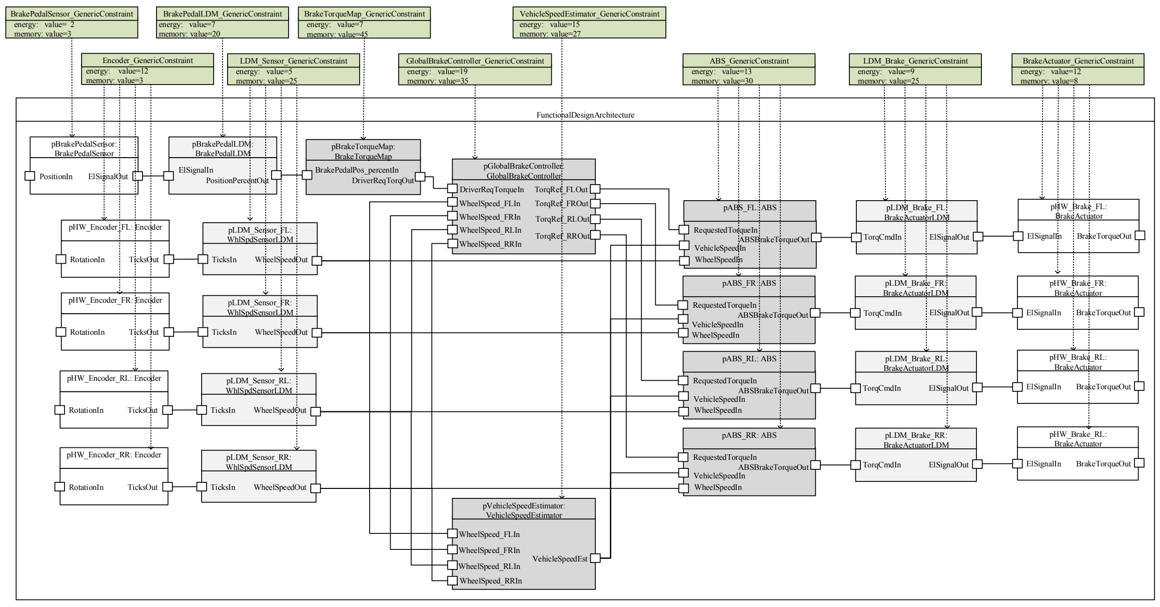


Figure 5. An overview of the Brake-by-Wire (BBW) system and its resource allocation.

The ABS algorithm calculates the slip rate s using the following equation:

$$S = (V - W \times R)/V,$$

where V is the vehicle speed, W is wheel speed, and R is the wheel radius. This coefficient has a nonlinear relationship with the slip rate: When s is increasing, the friction coefficient is also increasing, and its peak value is reached when s is 0.2. Further increase in s reduces the wheel friction coefficient. In this case, if s is greater than 0.2 the brake is released.

To exemplify the transformation of the BBW system modeled in EAST-ADL, we show a small-case example in Figure 6 that depicts the translation of the brake pedal sensor FunctionPrototype into a network of two synchronized PTA. For each component, the energy is calculated according to the rate of consumption monitored in the interface PTA ($pBrakePedalSensor\_e' == 2$), and a constant value is used in the behavior automaton ($pBrakePedalSensor\_e = pBrakePedalSensor\_e + 0.1$). The energy consumed by the entire system is computed using the sum of the energy consumed by each component.

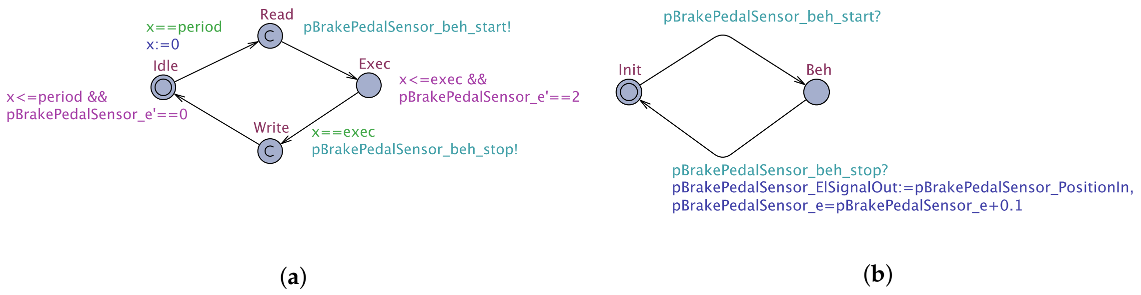

x==period
x:=0
Read
pBrakePedalSensor_beh_start!
Idle
Exec
x<=exec &&
pBrakePedalSensor_e'==2
x<=period &&
pBrakePedalSensor_e'==0
Write
x==exec
pBrakePedalSensor_beh_stop!
(a)
pBrakePedalSensor_beh_start?
Init
Beh
pBrakePedalSensor_beh_stop?
pBrakePedalSensor_ElSignalOut:=pBrakePedalSensor_PositionIn,
pBrakePedalSensor_e=pBrakePedalSensor_e+0.1
(b)

Figure 6. The network of Priced Timed Automata (PTA) for the brake pedal sensor component FunctionPrototype showing the transformed EAST-ADL architectural interface and behavior. (**a**) The interface timed automaton for the brake pedal sensor component. (**b**) The behavior timed automaton for the brake pedal sensor component.

4.2. Experimental Evaluation

We performed an experimental evaluation considering the creation of energy-aware mutations and model testing for the BBW system. We collected data for the following metrics: Generation time as a proxy for test efficiency, mutation score as a proxy for fault detection and the number of selected test cases as a measure of model testing cost reduction. In order to calculate the fault detection score, each test suite is executed on the faulty versions of the original EAST-ADL model to evaluate whether it detects the injected energy faults or not.

4.2.1. Test Suite Generation Results

As input for test generation, the method requires a standard EAST-ADL model. The generation stops searching for test inputs when it achieves the number of simulation runs. The MATS tool automatically runs the test cases on the BBW model and compares the expected outputs with the actual ones. We used different number of simulations (i.e., 25, 50, 100, 200, 400, 1000) to assess the test generation efficiency and effectiveness. As simulation time we used 64, value based on the BBW model and its full system execution and the calculated end-to-end deadline. In addition, we use 0.05 as the sample size for detecting the differences in the energy signal and 5 as the threshold delta. These values are selected based on our experience with verifying and analyzing the BBW system, as this is a realistic scenario and the values show significant differences in the energy consumption upon manual inspection of traces.

In Table 2, we present the results of applying model testing to the BBW case. As mentioned previously, in this experiment, we assume that the time is bounded to 64 time units. Table 2 lists—for each test suite and query to be checked—the time for test generation, as well as the mutation score achieved by each test suite. Regarding the generation of simulations for energy consumption, UPPAAL SMC is able to generate test cases between 17.2 s for 50 simulations and 563.4 s for 1000 simulations of the model.

In addition, to create a test suite demonstrating the worst-case energy consumption, we can use UPPAAL SMC's capacity to generate the maximum energy value. In this case, we generate simulations of the system over 200 runs, trying to maximize the energy during these simulations, with the query E[t<=64, 200](**max** : energy). The mean value generated by UPPAAL SMC is 447.2 energy units. Using this estimation, we use can use feasibility analysis for obtaining the probability for the energy value to remain within the threshold. For example, on the BBW system we are able to demonstrate, after 86 runs, that the energy consumption is lower than 447.2 with a 0.9 probability, and a confidence of 0.9. Given these results, an engineer can select test cases from the generated test suites showing the worst-case energy consumption and which can be executed on the actual system.

These results show the applicability of statistical model checking, for model testing the energy consumption using real architectural descriptions from an industrial system. We conclude that we have obtained experimental evidence that this is an efficient method for model testing applied on a real-world embedded system using its energy consumption information at the EAST-ADL architectural level. These results suggest that model testing is computationally inexpensive when used for mutation testing.

Table 2. Overall results showing the efficiency of the energy consumption test generation and mutant detection score for all generated test suites. In addition, we show the results of the test case (TC) selection based on mutation analysis.

Test Suite	SMC Query	Generation Time (s)	Mutation Score (%)	Selected TCs | Total TCs
TS1	simulate 25[<=64]{inputs[], energy}	17.2	30%	8|25
TS2	simulate 50[<=64]{inputs[], energy}	36.4	57%	15|50
TS3	simulate 100[<=64]{inputs[], energy}	72.1	75%	42|100
TS4	simulate 200[<=64]{inputs[], energy}	137.8	87%	59|200
TS5	simulate 400[<=64]{inputs[], energy}	277.5	100%	123|400
TS6	simulate 1000[<=64]{inputs[], energy}	563.4	100%	130|1000

4.2.2. Fault Seeding and Mutation Detection Results

The fault seeding procedure, results in 223 mutations (i.e., 50 ERO-based mutants, 48 PRO-based mutants, 50 ETRO-based mutants and 25 mutants for each CRO, CIO and TRO-based mutant operators). All mutants are versions of the original EAST-ADL model containing a single fault (i.e., each fault assumes one change in the system). All 223 faults are the result of applying all mutation operators. On each of the faulty versions and their original counterparts, the generated test suites are executed so that a fault detection score can be calculated. A mutant is deemed to be detected by a test suite if at some time points the energy values vary significantly. This is performed in order to increase the likelihood of detecting if the energy varies considerably from the expected result. Based on our experience in verifying and analyzing the BBW system, we set the threshold to 5 energy units.

The results in Table 2 show that for all test suites (i.e., TS1 to TS6) the achieved mutation score ranges from 30% for a test suite with 25 test cases to 100% for TS5 and TS6. Our results suggest that for test suites containing over 400 test cases all mutants are detected. Test suites TS5 and TS6 are assumed to be good at detecting faults in comparison to the other generated test suites. Even so, executing all 400 test cases is costly when performing testing on the actual BBW system and not all test cases contribute directly to the overall mutation score. This is extremely expensive from a computational point of view when considering using model testing on a complex embedded system. Therefore, we use the results of mutation analysis when executing test cases on the actual model to find the minimum number of test cases achieving the overall mutation score. In Table 2 we show the number of selected test cases out of the total number of test cases generated for each test suite (e.g., a subset of eight test cases can achieve a 30% mutation score). For all test suites, regardless of the number of generated test cases, one can reduce the number of test cases achieving the desired mutation score by over 40%. For TS5, 123 test cases are needed for achieving 100% mutation coverage with the rest of the test cases in TS5 not improving the overall score. This shows, that some test cases overlap in their exercised behavior of the BBW system and an improved generation strategy is needed to select the necessary test cases during model testing.

5. Validity Evaluation

Here we present a validity evaluation using the guidelines proposed by Runeson and Höst [26].

5.1. Construct Validity

Proper construct validity investigates the phenomenon that the researchers intended to study. The development of the method and the design of our experimental evaluation was based on certain assumptions. We have seeded energy-aware mutants automatically to calculate the ability of the selected test cases to detect energy errors. This process is carried out before test cases are generated in order to avoid a potential bias. It is possible that a larger number of naturally-occurring faults would yield different results. Adding real faults from previous projects should be employed in order to control the results more objectively early in the development process.

5.2. Internal Validity

For an experimental study as ours, internal validity relates to how credible the testing results and the mutation detection are. Detection of faults is based on a threshold of the energy budget and the time points selected to check the difference in the signal. This requirement is specific to the system and will not be enough to draw any strong conclusions. The effectiveness of this criterion depends on the energy difference interpretation and would clearly differ from one system to another. Because differences are characterized by the features of the signal shapes, we have checked at certain points in time whether the energy values differ substantially. This is a realistic situation with test engineers likely to find faults based on the measured energy consumption and the manual visual inspection.

5.3. External Validity

External validity relates to the study generalizability. Our method aims at designing and selecting a proper test suite based on a generic evaluation of the architectural model and the mutant detection score to reveal energy-related problems. However, unlike functional testing, which can use various metrics (e.g., code coverage, input space partitioning) for test generation, mutation testing for energy consumption is not as well studied. There is a need to develop and evaluate metrics capturing aspects of test effectiveness of energy consumption (such as those suggested in this paper) that can be used for test generation and selection.

6. Related Work

Recently, there has been a growing interest [27] in developing testing techniques focusing on architectural designs in software engineering. Testing based on software architectures has been explored in a considerable amount of work [28–32], leading to contributions in system and integration testing [28–30], criteria for architectural-based testing [31], and regression test selection [32]. Jiang et al. [33] compared several techniques that are used in performance and load testing of software intensive systems. For example, Zhang et al. [34,35] proposed the use of load testing of timing and resource requirements using system-level models. Compared to this work, we focus on energy consumption and we provide an efficient and effective model testing method; this aspect has received little attention in the literature.

Testing software based on extra-functional properties at the architectural level has received less attention [7,36–40] than the functional testing of such models. In our previous work [7], we explored the use of statistical model checking for analysis and performance testing of EAST-ADL models and used manually injected faults to measure test effectiveness. In this work, we build upon these results and consider how to automatically select and generate test suites for testing the energy performance of a system based on its architectural model and mutation testing.

One of the initial papers on non-functional testing [36] used a software architecture for selecting the parameters that directly influence the system performance. Among the approaches for non-functional test generation, only a few [41–44] consider robustness and performance aspects. Nebut et al. [41] and Shaukat et al. [42] proposed methods for automatic test generation that support robustness goals expressed in UML models. In contrast to these studies, our approach is tailored to an architectural language, which is an emerging notation in the automotive domain. We automatically select test cases that cover energy-aware injected faults on the model level early-on in the development process. In addition, in this paper we focus on selecting test cases for testing the performance of an existing Brake-By-Wire system by using the energy consumption information encoded in the architectural model.

7. Conclusions

In this paper we outline a method for testing EAST-ADL architectural models using energy-aware mutations. Furthermore this method uses UPPAAL SMC and MATS approaches to select test suites that contribute to the overall mutation score. The method makes use of energy requirements as expressed in EAST-ADL architectural models, transforms these requirements into priced timed automata together with the component interfaces, and uses statistical model checking to identify relevant test cases. We use simulations to create test suites containing input parameters and energy signals. We select test cases that maximize the mutation score. An experimental evaluation of this method, using a Brake-by-Wire system provided by Volvo Group Trucks Technology in Sweden, indicates that model testing of energy consumption is applicable for the automatic generation and execution of test suites at architectural level. The evaluation indicates that this method of creating test suites is efficient in terms of generation time. In this study, we proposed to evaluate the fault detection capability of these test suites by seeding modeling errors for energy consumption and altering the level of energy consumption over time. Our results suggest that an approach that selects test cases showing diverse energy consumption patterns can increase the fault detection ability.

Future work aims to extend our approach to generate tests for other types of resources and to apply it more thoroughly to real industrial cases to demonstrate its strengths and limitations by using naturally occurring faults.

Author Contributions: The first two authors contributed equally to the research, approach, study design, analysis and reporting of the research work. In addition, the first author contributed with tool development and data collection. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Swedish Research Council (VR) through the "Adequacy—based testing of extra-functional properties of embedded systems" project. This work is partially funded from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No. 737494, The Swedish Innovation Agency, Vinnova (MegaM@Rt2 and XIVT), as well as by the Swedish Knowledge Foundation (KKS), within the DPAC (Dependable Platforms for Autonomous Systems and Control) research profile.

Acknowledgments: We would like to thank Jonatan Larsson for his support in using the MATS tool and Raluca Marinescu for her valuable comments on this work.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Pretschner, A.; Broy, M.; Kruger, I.H.; Stauner, T. Software engineering for automotive systems: A roadmap. In Proceedings of the IEEE Computer Society on 2007 Future of Software Engineering, Minneapolis, MN, USA, 23–25 May 2007; pp. 55–71.
2. Hammond, J.; Rawlings, R.; Hall, A. Will it work?[requirements engineering]. In Proceedings of the IEEE International Symposium on Requirements Engineering, Toronto, ON, Canada, 27–31 August 2001, pp. 102–109.
3. Barroso, L.A.; Hölzle, U. The case for energy-proportional computing. *Computer* **2007**, *40*. [CrossRef]
4. Briand, L.; Nejati, S.; Sabetzadeh, M.; Bianculli, D. Testing the untestable: model testing of complex software-intensive systems. In Proceedings of the 38th International Conference on Software Engineering Companion, Austin, TX, USA, 14–22 May 2016; pp. 789–792.
5. González, C.A.; Varmazyar, M.; Nejati, S.; Briand, L.C.; Isasi, Y. Enabling model testing of cyber-physical systems. In Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, Copenhagen, Denmark, 14–19 October 2018; pp. 176–186.
6. Blom, H.; Lönn, H.; Hagl, F.; Papadopoulos, Y.; Reiser, M.O.; Sjöstedt, C.J.; Chen, D.J.; Tagliabò, F.; Torchiaro, S.; Tucci, S. EAST-ADL: An Architecture Description Language for Automotive Software-Intensive Systems. *EAST-ADL White Paper* **2013**, *1*.
7. Marinescu, R.; Enoiu, E.; Seceleanu, C.; Sundmark, D. Automatic Test Generation for Energy Consumption of Embedded Systems Modeled in EAST-ADL. In Proceedings of the 2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), Toyko, Japan, 13–17 March 2017; pp. 69–76.
8. DeMillo, R.A.; Lipton, R.J.; Sayward, F.G. Hints on test data selection: Help for the practicing programmer. *Computer* **1978**, *11*, 34–41. [CrossRef]
9. Behrmann, G.; Fehnker, A.; Hune, T.; Larsen, K.; Pettersson, P.; Romijn, J.; Vaandrager, F. Minimum-Cost Reachability for Priced Time Automata. In *Hybrid Systems: Computation and Control*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2001.
10. Bulychev, P.; David, A.; Larsen, K.G.; Mikučionis, M.; Poulsen, D.B.; Legay, A.; Wang, Z. UPPAAL-SMC: Statistical Model Checking for Priced Timed Automata. In Proceedings of the Workshop on Quantitative Aspects of Programming Languages and Systems, Tallinn, Estonia, 31 March–1 April 2012.
11. Shaw, M.; Garlan, D. *Software Architecture: Perspectives on An Emerging Discipline*; Prentice Hall Englewood Cliffs: Englewood Cliffs, NJ, USA, 1996; Volume 1.
12. Ammann, P.; Offutt, J. *Introduction to Software Testing*; Cambridge University Press: Cambridge, UK, 2008.
13. Malavolta, I.; Lago, P.; Muccini, H.; Pelliccione, P.; Tang, A. What industry needs from architectural languages: A survey. *IEEE Trans. Softw. Eng.* **2013**, *39*, 869–891. [CrossRef]
14. Just, R.; Jalali, D.; Inozemtseva, L.; Ernst, M.D.; Holmes, R.; Fraser, G. Are Mutants a Valid Substitute for Real Faults in Software Testing? In *International Symposium on Foundations of Software Engineering*; ACM: New York, NY, USA, 2014.
15. Black, P.E.; Okun, V.; Yesha, Y. Mutation operators for specifications. In Proceedings of the ASE 2000. Fifteenth IEEE International Conference on Automated Software Engineering, Grenoble, France, 11–15 September 2000; pp. 81–88.
16. Aichernig, B.K.; Salas, P.A.P. Test case generation by OCL mutation and constraint solving. In Proceedings of the Fifth International Conference on Quality Software (QSIC'05), Melbourne, Australia, 19–20 September 2005; pp. 64–71.
17. Aichernig, B.K.; Jöbstl, E.; Tiran, S. Model-based mutation testing via symbolic refinement checking. *Sci. Comput. Program.* **2015**, *97*, 383–404. [CrossRef]
18. Lindström, B.; Offutt, J.; Sundmark, D.; Andler, S.F.; Pettersson, P. Using mutation to design tests for aspect-oriented models. *Inf. Softw. Technol.* **2017**, *81*, 112–130. [CrossRef]
19. Kang, E.Y.; Enoiu, E.P.; Marinescu, R.; Seceleanu, C.; Schobbens, P.Y.; Pettersson, P. A Methodology for Formal Analysis and Verification of EAST-ADL Models. *Reliab. Eng. Syst. Saf.* **2013**, *120*. [CrossRef]
20. Marinescu, R.; Kaijser, H.; Mikučionis, M.; Seceleanu, C.; Lönn, H.; David, A. Analyzing industrial architectural models by simulation and model-checking. In *International Workshop on Formal Techniques for Safety-Critical Systems*; Springer: London, UK, 2014; pp. 189–205.

21. Bulychev, P.; David, A.; Larsen, K.G.; Legay, A.; Li, G.; Poulsen, D.B. *Rewrite-Based Statistical Model Checking of WMTL*; Runtime Verification; Springer: London, UK, 2013.
22. Marinescu, R.; Enoiu, E.; Seceleanu, C. *Statistical Analysis of Resource Usage of Embedded Systems Modeled in EAST-ADL*; VLSI Symposium; IEEE: Piscataway, NJ, USA, 2015; pp. 380–385.
23. Aichernig, B.K.; Lorber, F.; Ničković, D. Time for mutants—model-based mutation testing with timed automata. In *International Conference on Tests and Proofs*; Springer: London, UK, 2013, pp. 20–38.
24. Seceleanu, C.; Vulgarakis, A.; Pettersson, P. REMES: A Resource Model for Embedded Systems. In Proceedings of the International Conference on Engineering of Complex Computer Systems, Potsdam, Germany, 2–4 June 2009.
25. Larsson, J. *Automatic Test Generation and Mutation Analysis using UPPAAL SMC*; Bachelor of Science Thesis Report; MDH Diva: Ascona, Switzerland, 2017.
26. Runeson, P.; Höst, M. Guidelines for conducting and reporting case study research in software engineering. *Empir. Softw. Eng.* **2009**, *14*, 131. [CrossRef]
27. Bertolino, A.; Inverardi, P.; Muccini, H. Software architecture-based analysis and testing: A look into achievements and future challenges. *Computing* **2013**, *95*, 633–648. [CrossRef]
28. Abdurazik, A.; Jin, Z.; White, L.; Offutt, J. Analyzing software architecture descriptions to generate system-level tests. In Proceedings of the Workshop on Evaluating Software Architectural Solutions, Irvine, CA, USA, 9 May 2000.
29. Bertolino, A.; Inverardi, P. Architecture-based software testing. In *International Software Architecture Workshop*; ACM: New York, NY, USA, 1996; pp. 62–64.
30. Richardson, D.J.; Wolf, A.L. Software testing at the architectural level. In *International Software Architecture Workshop and International Workshop on Multiple Perspectives in Software Development*; ACM: New York, NY, USA, 1996, pp. 68–71.
31. Jin, Z.; Offutt, J. Deriving tests from software architectures. In Proceedings of the International Symposium on Software Reliability Engineering, Hong Kong, China, 27–30 November 2001, pp. 308–313.
32. Harrold, M.J. Architecture-based regression testing of evolving systems. In Proceedings of the International Workshop on the Role of Software Architecture in Testing and Analysis, Marsala, Italy, 30 June–3 July 1998.
33. Jiang, Z.M.; Hassan, A.E. A Survey on Load Testing of Large-Scale Software Systems. *IEEE Trans. Softw. Eng.* **2015**, *41*, 1091–1118. [CrossRef]
34. Zhang, J.; Cheung, S.C. Automated test case generation for the stress testing of multimedia systems. *Softw. Pract. Exp.* **2002**, *32*, 1411–1435. [CrossRef]
35. Zhang, J.; Cheung, S.C.; Chanson, S.T. Stress testing of distributed multimedia software systems. In *Formal Methods for Protocol Engineering and Distributed Systems*; Springer: London, UK, 1999; pp. 119–133.
36. Weyuker, E.J.; Vokolos, F.I. Experience with performance testing of software systems: issues, an approach, and case study. *IEEE Trans. Softw. Eng.* **2000**, *26*, 1147. [CrossRef]
37. Denaro, G.; Polini, A.; Emmerich, W. Early performance testing of distributed software applications. In *ACM SIGSOFT Software Engineering Notes*; ACM: New York, NY, USA, 2004; Volume 29, pp. 94–103.
38. Jiang, Z.M.; Hassan, A.E.; Hamann, G.; Flora, P. Automated performance analysis of load tests. In Proceedings of the International Conference on Software Maintenance, Edmonton, AB, Canada, 20–26 September 2009; pp. 125–134.
39. Grechanik, M.; Fu, C.; Xie, Q. Automatically finding performance problems with feedback-directed learning software testing. In Proceedings of the International Conference on Software Engineering (ICSE), Zurich Switzerland, 2 June 2012; pp. 156–166.
40. Marinescu, R.; Saadatmand, M.; Bucaioni, A.; Seceleanu, C.; Pettersson, P. A model-based testing framework for automotive embedded systems. In Proceedings of the 2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications, Verona, Italy, 27–29 August 2014; pp. 38–47.
41. Nebut, C.; Fleurey, F.; Le Traon, Y.; Jezequel, J.M. Automatic test generation: A use case driven approach. *IEEE Trans. Softw. Eng.* **2006**, *32*, 140–155. [CrossRef]
42. Ali, S.; Briand, L.C.; Hemmati, H. Modeling robustness behavior using aspect-oriented modeling to support robustness testing of industrial systems. *Softw. Syst. Model.* **2012**, *11*, 633–670. [CrossRef]

43. Yue, T.; Ali, S. Bridging the gap between requirements and aspect state machines to support non-functional testing: industrial case studies. In *European Conference on Modelling Foundations and Applications*; Springer: London, UK, 2012; pp. 133–145.
44. Garousi, V.; Briand, L.C.; Labiche, Y. A UML-based quantitative framework for early prediction of resource usage and load in distributed real-time systems. *Softw. Syst. Model.* **2009**, *8*, 275–302. [CrossRef]