Automatic Test Generation for Energy Consumption of Embedded Systems Modeled in EAST-ADL

Raluca Marinescu*, Eduard Enoiu*, Cristina Seceleanu*, and Daniel Sundmark*
*Mälardalen University, Västerås, Sweden, <firstname.lastname>@mdh.se

***Abstract*—Testing using architectural design models is intended to determine if the realized system meets its specification, and works as a whole in terms of computational components and their interactions. The growing complexity of embedded systems requires new techniques that are able to support testing of extra-functional requirements, like energy usage of components and systems, which is very necessary in order to obtain valid implementations. In this paper, we show how architectural models described in the EAST-ADL architectural language can also be used for testing the energy consumption of embedded systems, after transforming them into networks of formal models called priced timed automata. Assuming an EAST-ADL model annotated with energy consumption information, we show how to automatically generate energy-aware test cases based on statistical model checking (SMC) of the resulting network of priced timed automata. We automate the generation of executable test cases with UPPAAL SMC, using a test strategy based on several random simulation runs of the system. By seeding the original formal model with a set of energy-consumption related faults, we are able to carry out fault detection analysis. We apply this technique on a Brake-by-Wire system from the automotive domain, and evaluate it in terms of efficiency and model fault detection.**

I. INTRODUCTION

The use of embedded systems is ubiquitous. For instance, using embedded systems in the automotive industry allows for the implementation of complex automotive features, such as cruise control, automatic braking, and stability control [24]. Such features also increase the complexity and heterogeneity of the entire system [15], and significant problems could arise during the integration of new sub-systems due to the emergent behavior of the whole system. For example, a change in the software sub-system or the replacement of a hardware component can affect the system's latency, energy consumption and memory utilization [24]. In such cases, it is not enough to show the functional correctness of each sub-system's behavior, but one also needs to provide evidence that the system meets its extra-functional requirements, such as real-time performance and resource consumption.

Given the growing demand for low-energy computing in embedded systems [5], the energy consumption of software is becoming an increasingly important issue in designing such systems. Therefore, testing the system's behavior with respect to not exceeding its provided energy budget, the so-called feasibility tests, as well as testing for the worst-case energy consumption of components and system are very important for ensuring the quality of service of the embedded system, and estimating its performance.

To facilitate reasoning about resource consumption of embedded systems at high levels of abstraction, architectural description languages such as EAST-ADL[1] [9] enable the representation of both hardware and software elements, as well as related extra-functional information (e.g., timing properties, triggering information, resource consumption), by annotating the model accordingly. If we assume energy as the resource of interest, it is not well studied how energy consumption annotations of EAST-ADL models can be used to create performance tests for feasibility and worst-case energy consumption of components.

Since load testers might have limited resources to manually create test suites for all performance scenarios, we propose a method for automatically generating test suites by selecting the nominal and potentially troublesome simulations of an embedded system modeled in EAST-ADL after transforming it into a network of priced timed automata [6]. Specifically, we select test suites based on random simulations showing the nominal energy consumption and the most expensive energy cost based on a statistical analysis of the system using UPPAAL SMC [11], the statistical extension of the UPPAAL model checker. We show how to seed faults in the EAST-ADL model and assess the energy-related fault detection capability of each generated test suite.

To illustrate the efficiency of our method, we carry out a pilot evaluation, using a Brake-by-Wire industrial prototype system. This system is modeled in EAST-ADL architectural language annotated with energy resources. Our results show that the proposed method is efficient in terms of generation time, and that an approach that selects test suites showing divergent energy consumption from the expected result can increase the fault detection scores. Our observations regarding fault detection could allow an industrial designer to gain deeper understanding into the system's resource-usage behavior, and consequently adjust and optimize both software and hardware designs accordingly.

The remainder of the paper is organized as follows. In section II we overview the preliminaries needed to comprehend our contribution, that is, the concept of architecture-based testing, EAST-ADL language, UPPAAL SMC and priced timed automata, respectively. The main contribution of the paper, that is, our approach for creating energy-aware tests from annotated EAST-ADL models is described in section III, and its

[1]EAST-ADL stands for Electronic Architecture and Software Tools-Architecture Description Language.

application on the Brake-by-Wire system and the experimental results are presented in section IV. We discuss our conclusions and present lines of future work in section VII.

II. Background

In this section, major aspects of architecture-based testing, the EAST-ADL language and UPPAAL SMC are discussed. The presented aspects are put into the context of the contribution of this study.

A. Architecture-Based Software Testing

Architectural design models are created during software development by choosing components and connectors representing the whole software system and its high-level structure [26]. Testing using architectural designs as input to the creation of tests is intended to check if the system meets its architectural specifications in terms of the overall interactions among the system components. This type of testing is sometimes known as system testing [4] and its main purpose is to discover architectural design problems. Testing, in this case, may address such properties as functional correctness, real-time guarantees and performance [26]. According to an industrial survey [21], testing and analysis of extra-functional properties (e.g., energy, bandwidth and memory) are very important for practitioners when engineering embedded systems. In this study we focus on testing for energy consumption based on architectural design models. When testing for such performance properties at the software architecture level, architectural models are annotated with energy consumption requirements. The purpose of testing the energy consumption (e.g., feasibility or worst case energy consumption) at the architectural level is to find faults with the performance of an actually developed system in terms of its components and interactions.

B. EAST-ADL Architectural Language

EAST-ADL [9] is an AUTOSAR-compatible[2][1] architectural description language dedicated to the development of automotive embedded systems. The functionality of the system is defined at four levels of abstraction, as follows: (i) the *Vehicle Level*, the highest level of abstraction, describes the electronic features as they are perceived externally, (ii) the *Analysis Level* provides an abstract functional representation of the architecture, (iii) the *Design Level* provides a detailed functional representation of the architecture, together with the allocation of these elements onto the hardware platform, and (iv) the *Implementation Level* provides the implementation of the system using AUTOSAR elements.

At each abstraction level, the system model uses components, each a *FunctionType*, which describe the functional elements of the system. The *FunctionType* has: (i) ports that receive and provide data, respectively, (ii) a trigger, either time-based or event-based, and (iii) an associated behavior. Each of these components is instantiated as one or more of type *FunctionPrototype*, which are connected to provide the system model. The execution of each *FunctionPrototype* is based on the "*read-execute-write*" semantics, and the associated behavior can be defined using different notations and tools (e.g., Simulink or UPPAAL PORT timed automata [20]). The work proposed in this paper targets the information encoded at the *Design Level*, since it depicts both the *Functional Design Architecture* (FDA) and *Hardware Design Architecture* (HDA), making it possible to specify extra-functional properties such as allocation, efficiency, reuse. The model can be extended with a *GenericConstraint* annotation, which allows the system designer to specify various extra-functional properties, such as energy consumption or memory utilization.

[2]AUTOSAR stands for AUTomotive Open System ARchitecture developed by manufactures as a standard in the automotive domain.

C. UPPAAL SMC and Priced Timed Automata

UPPAAL SMC [11] is an extension of UPPAAL, enabling the analysis of different performance properties of networks of priced timed automata with stochastic semantics. Statistical model-checking generates stochastic simulations to estimate probabilities and probability distributions over time with given confidence levels, so the technique scales better than exact symbolic model-checking.

Priced timed automata (PTA) are extensions of timed automata with cost variables that can evolve at integer rates (also $\neq 1$) and are used in this paper to capture the resource usage. In this paper, we restrict the models to natural number rates. The resource usage is modeled via a function $P : (L \cup E) \rightarrow \mathbb{N}$, where L is a finite set of locations, and E is the set of edges of the PTA model, which assigns costs to both locations and edges. A network of PTA (NPTA) can be expressed as a composition of n PTA over clocks and actions; the PTA synchronize on send-receive actions (i.e., *send* $b!$ is complementary to *receive* $b?$) and can use shared variables in guards that are Boolean conditions enabling the execution of edges).

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, $I : L \rightarrow B(X)$ 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 $P : (L \cup E) \rightarrow \mathbb{N}$ 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

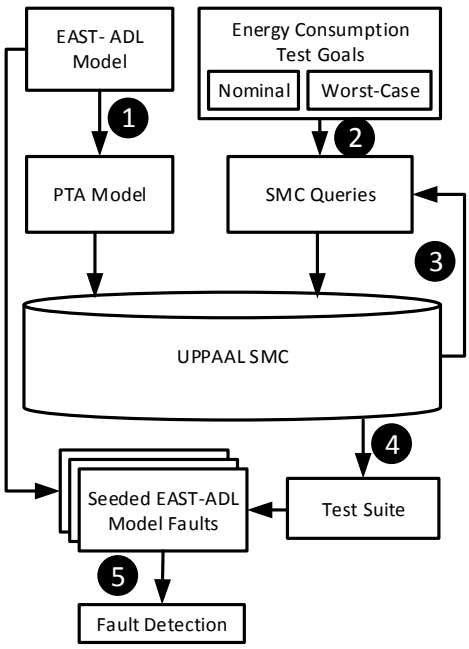


Fig. 1: Overview of the test suite generation and evaluation method for energy consumption based on EAST-ADL architectural models.

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 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 $\Sigma_{i=1}^{n} p_i$.

A network of PTA $A_1 \parallel ... \parallel A_n$ can be expressed as a composition of n PTA over X and Act, synchronizing on complementary actions and using shared variables that can be used in guards and transitions.

UPPAAL SMC uses an extension of Weighted Metric Temporal Logic (WMTL) [10] to carry out hypothesis testing, among other properties, which checks if the probability to reach state ϕ within cost $x \leq C$ is greater or equal to a certain threshold $p : Pr(\Diamond_{x \leq C} \phi) \geq p$.

III. Creating Energy-aware Tests for East-adl Models

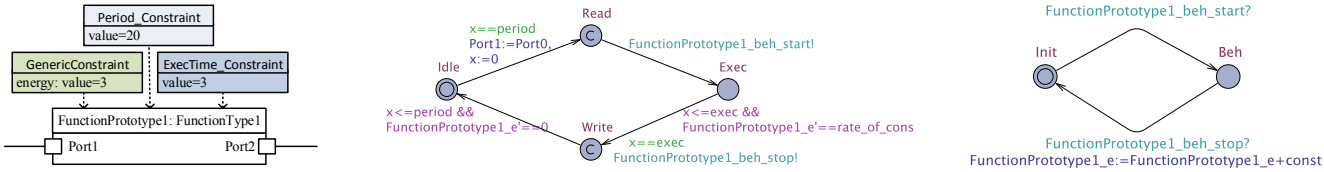
In this section, we describe an approach to automatically generate test suites for testing the energy consumption using an EAST-ADL system architectural model. Our test generation approach aims to use energy consumption goals (e.g, nominal, worst-case energy consumption) to automatically select test suites using several random simulations of the system.

Overall, the approach is composed of the following steps, mirrored in Figure 1:

1) EAST-ADL TO PTA TRANSFORMATION. This first step (described in detail in Section III-A) shows the transformation of an EAST-ADL model containing energy consumption annotations into a PTA model ready to be used by UPPAAL SMC for test-case generation.
2) TEST GOAL TRANSFORMATION. The second step (described in detail in Section III-B) shows how a test query is obtained. This is a property expressible as a simulation property used by UPPAAL SMC.
3) TEST SUITE SELECTION. The third step (described in Section III-C) shows how we select test suites based on the needed resource analysis performed on the PTA model.
4) TEST SUITE GENERATION. The fourth step (described in Section III-D) requires the use of the UPPAAL SMC to generate a set of test cases satisfying the test goal by using the SMC's ability to show simulation traces.
5) TEST SUITE EVALUATION. The fifth step (described in Section III-E) shows how test oracles are used to automatically assess the energy-related fault-revealing ability of the generated test suites.

A. EAST-ADL to PTA Transformation

We represent the architectural elements in EAST-ADL by an automatic transformation into PTA. Each *FunctionPrototype* is transformed into a network of two synchronized automata (as shown in Figure 2): an interface automaton, dedicated to the elements of the EAST-ADL component interface, and a behavior automaton, used to model the behavior of the EAST-ADL components. Each $FunctionPrototype$ is defined as an automaton with four locations: (i) $Idle$, (ii) a $Read$ location that allows the update of the variables according to the values on the input ports, independent of other computations, (iii) an $Exec$ location that triggers the $Behavior$ that models the desired behavior of $FunctionPrototype$, and (iv) a $Write$ location that allows the update of the output ports according to the values of the computed internal variables, respectively, independent of other computations. The triggering of each interface is based on the triggering $Trigg$ associated to the EAST-ADL $FunctionPrototype$. Other resource annotations EC, e.g., the energy consumption in the *GenericConstraint* annotation, are included in the behavior of the TA model and a monitor automaton is used to measure the resource consumption. Concretely, we use a monitor automaton that contains all the resource annotations of the EAST-ADL model, including the energy consumption. The monitor is a loop-free PTA that follows the execution of the system, which is achieved by employing the already existing synchronization channels *FunctionPrototype_beh_start* and *FunctionPrototype_beh_stop*. Assuming an architectural model consisting of one *FunctionPrototype* only, we annotate the monitor with a continuous resource, that is, energy, whose consumption is increasing with time. The energy is consumed from the time when data is read from the input ports until the function block writes the data to the output ports. The *GenericConstraint* allows the EAST-ADL model to be annotated with its energy



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

Fig. 2: An example of a transformed EAST-ADL component using a generic interface timed automaton and a behavior template.

consumption. The consumption of a continuous, non-referable resource (e.g., energy) c is computed as $c = n \times t + s$, where $n \in Z$ the rate of consumption over time t, and $s \in Z$ is a constant. Figure 2 depicts the results of the transformation for a generic EAST-ADL *FunctionPrototype*. For more details, we refer the reader to our previous work [22].

B. Test Goal Transformation

Extra-functional aspects contained in the EAST-ADL, like energy consumption, are often more difficult to test than functional properties. Embedded systems often require the consumption of large amounts of energy. Irregular and heavy energy usage could lead to inadequate system functionality to keep essential components running. Utilizing the energy consumption annotated in EAST-ADL, an estimation of the dedicated energy budget can be computed. If the actual energy consumption of the system does not exceed its energy budget, the system can complete its execution. Otherwise, the system has exhausted its given budget. In this study, we address the goal of analyzing the energy consumption over time. Consequently, we focus on test queries providing the simulation of the nominal and worst-case energy consumption of a system modeled in EAST-ADL. Our technique provides simulation traces over a predefined number of runs of the system model. Simulation can be formulated 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 expressions to be monitored.

Even though the core idea of UPPAAL SMC is to provide statistical results based on a series of simulations of the system, it is desirable to use the values of the input parameters and energy during individual simulations of the system as test cases. The simulation is dependent on the number of runs of the model (n), and the upper time limit for the number of simulation runs ($bound$).

C. Test Suite Selection

The previous phase can simulate the energy consumption from each individual run of the model. This step evaluates the overall energy performance of each simulation and pinpoints the simulation with the worst-case energy consumption. The analysis using UPPAAL SMC can return a maximal value of the energy that will eventually reach a certain behavior in time. This problem is reduced to maximizing the energy cost function such that the following property is satisfied:

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

where $bound$ is the same time bound used for generating simulation traces, n gives the number of runs, and $energy$ is the energy cost to be evaluated. The worst-case energy consumption analysis computes, on the network of PTA, the cost of the simulation that will eventually reach a certain timed behavior. The maximum value of the energy obtained from the consumption analysis is used to identify a test suite showing the estimated worst-case energy consumption. Feasibility analysis is used at this point to verify if the energy usage stays within the maximum energy value provided by the worst-case energy consumption analysis. The verification is achieved by examining the probability distribution of the energy resource, which is formulated as a probability evaluation, as follows:

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

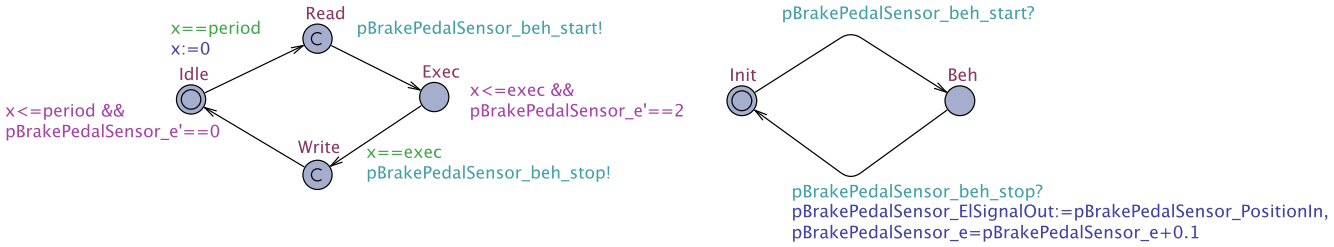
where $bound$ defines the time bound for the runs, and ψ can be of the form "Eventually q", where q is a state predicate. Based on this analysis we can select the necessary test suites. As the load tester might have limited time to test all scenarios in practice, we select the nominal and potentially troublesome simulations. Specifically, we select several random simulations showing the nominal energy consumption and another one which determines when the energy cost is the most expensive.

D. Test Suite Generation

We use the simulation traces to create executable test cases. Test cases are obtained by extracting from the simulations the input parameters and the energy values at each predefined time unit. In UPPAAL SMC, every input variable is described using signals. In this context, test suite generation is essentially signal generation using the simulations generated by UPPAAL SMC. Each test input is a vector of signals. Given that the search space of input signals is very large, we select the input signals evolving over a certain predefined time. We note that our input signal generation is geared towards the specific needs of the automotive domain where EAST-ADL is used, where the time-dependent behavior of the system is tested using ordered sequences of signals as input values.

E. Test Suite Evaluation

For fault-seeding purposes, one should model representative naturally-occurring faults related to the energy consumption.



(a) The interface timed automaton.

(b) The behavior timed automaton.

Fig. 3: The network of Priced Timed Automata (PTA) for the brake pedal sensor component *FunctionPrototype* showing the transformed EAST-ADL architectural interface and behavior.

Unfortunately, to the best of our knowledge, there is no accepted way to determine whether seeded faults for PTA models are representative. We suggest in this study to alter the PTA model in a small way, mimicking the energy cost consumption modeling errors. Given a PTA model, its faults are seeded by changing the rate of consumption over time or the energy constant.

A fault is considered to be detected by a test suite if the energy values differ drastically at certain time points in order to increase the probability of evaluating if the energy noticeably diverge from the expected result. Given that small deviations from the specified energy values can be acceptable, test engineers are likely to identify any fault when provided with substantial energy deviations. To measure the fault-revealing ability of a test suite, we use a quantitative measure of test verdict. Let be a test suite TS generated and executed for a given faulty model M, and let $E = e_1, ..., e_n$ be the set of energy signals obtained by running M for the test inputs in TS and sampled at n time points. Let $O = o_1, ..., o_n$ be the corresponding expected signals. We use a threshold value to check if the distance between each value of E and M at the same time points is larger than this budget threshold, respectively. If there is at least one energy value in O for which the energy values of E sufficiently deviates from the expected value then a tester could conclusively detect a fault. Otherwise, the faulty model is not detected by the test suite TS.

IV. A CASE STUDY ON A BRAKE-BY-WIRE SYSTEM

In order to evaluate the proposed automatic test generation technique, we apply it on an industrial prototype system provided by Volvo Group Trucks Technology, Sweden. We perform some preliminary experiments on a Brake-by-Wire (BBW) industrial prototype and evaluated the applicability (i.e., generation time) of creating test suites using the proposed automatic test generation for energy consumption. In addition, we investigate the energy-related fault detection ability of the generated test suites by using manually seeded faults. To facilitate fault-detection analysis, we begin by seeding a set of faults in the original model. For the creation of faults, we rely on energy consumption faults.

Concretely, we consider the original transformed PTA model and for each faulty model we execute the generated test suites and collect the simulation traces containing the energy values. In order to calculate the fault detection score, each test suite is executed on both the original model and its faulty counterpart. In case the energy results differ between the executions, the fault is considered to be detected.

A. Case Description

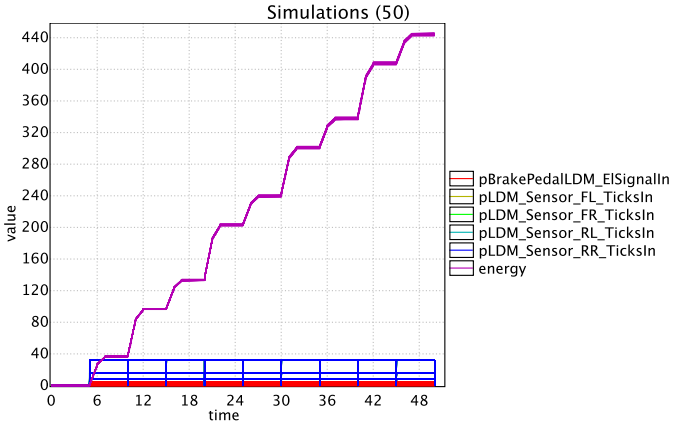
The BBW system is a braking system equipped with an Anti-Lock Braking (ABS) function, and without any mechanical connectors between the brake pedal and the brake actuators. A sensor attached to the brake pedal reads its position, which is used to compute the desired global brake torque. At each wheel, a sensor measures the speed of the wheel, which is used by the ABS algorithm together with the brake torque and the estimated vehicle speed to compute the actual brake torque that will be sent to the actuators. We have applied the transformation described in Section III-A on the BBW system modeled in EAST-ADL. Figure 3 depicts the transformation of the brake pedal sensor *FunctionPrototype* (part of the BBW system modeled in EAST-ADL) into a network of two synchronized PTA. The energy consumed by the component is calculated according to the rate of consumption modeled in the interface PTA ($pBrakePedalSensor\_e' == 2$), and the constant is modeled in the behavior automaton ($pBrakePedalSensor\_e = pBrakePedalSensor\_e + 0.1$). At any given time, the energy consumed by the system is calculated by adding the energy consumed by all of its components.

B. Manual Fault Seeding

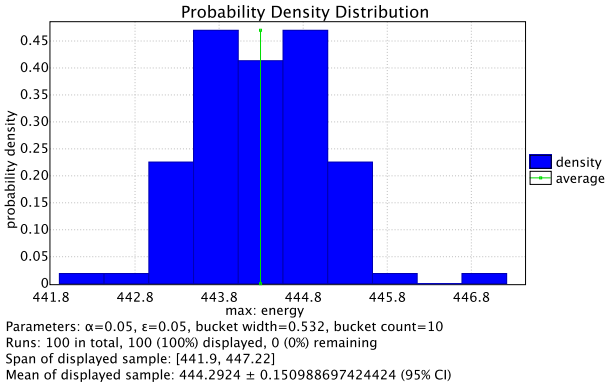
The fault seeding procedure, described in Section III-E, results in 5 faults (i.e., Fault 1 to 5), each being a version of the original EAST-ADL model containing a single fault (i.e., each fault assumes one or more changes in the model). These five faults are used in this study for characterizing potentially problematic energy consumption behavior. Each of the generated test suites is executed on each of the faulty versions and its original counterpart, so that a fault detection score can be calculated. A fault is considered to be detected

Resource	Test Goal Type	SMC Query	Time (s)	Runs
Energy	*Worst-Case Energy Consumption*	E[t<=50,100](**max** : energy)	11.8	100
	Feasibility Analysis	Pr[t<=50](<> energy<=447.22)	9.9	86
	Test Simulations Generation	simulate 50[<=50]{inputs[], energy}	7.2	50

TABLE I: Overall results showing the efficiency of the energy consumption analysis and test generation.



(a) Test simulations of energy consumption.



(b) Estimated energy probability distribution.

Fig. 4: Generation of simulations for energy consumption analysis showing nominal and worst-case test suites.

by a test suite if the energy values differ drastically at certain time points in order to increase the probability of evaluating if the energy noticeably diverge from the expected result. In our work, we set the threshold to 2 energy units. We choose this value based on our experience with verifying and analyzing the BBW system. Additional details on the case study (e.g, models and faults used) can be found at the study website created for storing the information needed for replication and reviewing[3].

C. Experimental Results and Discussion

In Table I, we present the overall results of our test-case generation method applied to the BBW system. In this experiment, we assume that the time is bounded to 50 time units in the simulations. This estimation is based on the analysis, using UPPAAL SMC, of the time needed for signals to propagate from the pedal sensors to the actuators in the BBW system. Table I lists–for each test goal type and query to be checked– the time used, as well as the number of simulation runs. Regarding the generation of simulations for energy analysis, UPPAAL SMC is able to find a solution in 7.2 seconds[4] within 50 simulations of the model (as shown in Figure 4a).

From these simulations we randomly select two test suites (T1 and T2) to partially show the nominal behavior of the simulated system. To generate a test suite showing the worst-case energy consumption, we use UPPAAL SMC's ability to compute the maximum expected value of the energy. For this, we simulate the system over 100 runs, trying to maximize the energy consumption, with the query E[t<=50, 100](**max** : energy). The mean value provided by UPPAAL SMC for the maximum consumption is 447.2 energy units. We record the distribution of the energy consumption over 100 runs, as shown in Figure 4b. Using this estimation, we use feasibility analysis to analyze the probability for the energy consumption to stay within the available energy threshold provided by the model. UPPAAL SMC is able to show that the energy consumption is smaller than 447.2 with a probability between 0.9 and 1, and a confidence of 0.9 after 86 runs of the model. Based on this analysis we select a third test suite (T3) from the generated simulations showing the worst-case energy consumption. From Table I we can observe that energy feasibility analysis and worst-case energy consumption analysis are computationally inexpensive when used for test-case generation for testing the energy consumption annotated in an EAST-ADL model. This shows how capable statistical model checking is, for testing the energy consumption of a realistic industrial system. We conclude that we have provided preliminary evidence that this is an efficient method for test generation for testing the energy consumption of a real-world embedded system modeled using EAST-ADL.

Regarding the fault detection in terms of detecting manually seeded faults, as shown in Table II, we focus on comparing all three test suites generated using our method. For all faults, the fault detection score obtained by T1 is higher than the

[3]We provide the models at www.testinghabits.org/autoenergy/

[4]Simulations ran on a computer with 1.8 Ghz processor and 8GB memory

TABLE II: Fault detection results for each generated test suite; X represents a fault being detected by a test suite and $-$ represents a fault not detected by a test suite.

Fault Detection\ Test Suite	$T1$	$T2$	$T3$
Fault 5	$-$	$-$	$-$
Fault 4	X	$-$	$-$
Fault 3	X	X	X
Fault 2	X	X	X
Fault 1	X	X	X
Fault Detection Score (%)	80%	60%	60%

one achieved by T2 and T3. It seems that Fault 4 is detected only by T1, while Fault 5 is not detected by any of the test suites. This can be explained by the fact that Fault 5 contains changes in components using less energy in each periodic execution. Fault 1, 2 and 3 are easily detected by all test suites. Since these three faults contain changes in components dissipating more energy per each execution cycle, the selected test suites easily detect the change in energy consumption. It is interesting to note that the test suite generated for showing the worst-case energy consumption is not able to detect two out of five faults. In our experiment, even though the test suite showing the worst-case energy consumption (i.e., T3) generated by UPPAAL SMC covers 60% of the seeded model faults, the energy values produced by this test suite on Fault 4 and 5 either do not deviate at all, or deviate just slightly from the oracle values, hence yielding very small deviation values. In contrast, the random selection of a test suite (i.e., T1) can result in an energy signal that is more distant from the oracle energy consumption value. It seems that an approach that selects test cases that yield diverse energy consumption can potentially increase the fault detection score.

V. THREATS TO VALIDITY

We have manually seeded energy-related faults to measure the fault detection capability of the selected tests. This has been performed prior to the generation of tests in order to avoid a potential knowledge bias. It is possible that a larger number of naturally-occurring faults or automatically-generated mutations would yield different results. Adding mutations should be employed in order to control the results more objectively.

The detection of faults is based on an energy budget threshold, and the selected time points for checking the signal difference. This criterion is case-study specific and is not sufficient to draw any strong conclusions. The effectiveness of this criterion depends on the definition of energy difference and would obviously differ from one context to another. As differences are characterized by signal shape features, we have checked if the energy values differ substantially at certain time points. This is a realistic situation with test engineers likely to identify faults based on visual inspection of the measured energy signal.

The approach presented in this study is focusing on designing and selecting a proper test suite for exposing resource-related problems based on analyzing the architectural model and the fault detection score. However, unlike functional testing, which can use various metrics (e.g., code coverage, mutation coverage) for test generation, resource-usage testing approaches are not as well studied. There is a need for establishing and evaluating metrics capturing performance and resource consumption aspects of test effectiveness that can be used for test generation and selection.

VI. RELATED WORK

Recently, there has been a growing interest [8] in developing testing and analysis techniques based on the architectural design of the system under test. Testing based on software architectures has been explored in a considerable amount of work, leading to contributions in the automatic generation of integration and system level tests [2], [7], [25], architectural-based testing criteria [19], and regression testing [16]. Testing software based on performance properties at the architectural level (also known as system level) has received less attention [27], [12], [18], [14] than the functional testing of such models. In this work, we directly consider how to select or generate test suites for testing the energy performance of a system based on its architectural model. One of the initial research papers on this topic [27] uses the software architecture of a system whose performance is used for selecting the parameters that directly influence the system performance. Among the numerous approaches proposed to generate tests, only a few [23], [3], [28], [13] are considering robustness and resource consumption. Nebut et al. [23] and Shaukat et al. [3] propose methods for automatic test generation that support robustness goals expressed in UML models. In contrast to these studies, our approach is based on the EAST-ADL architectural model, which is an emerging notation currently used in the automotive domain. In addition, we discuss in this study a related approach to selecting tests that we have found useful for testing the performance of an existing Brake-By-Wire system by using energy consumption found in the architectural model for generating tests.

Recently, Jiang et al. [17] compare current techniques that are used in performance and load testing. A few approaches [29], [30] have been proposed to automatically find problematic load tests which can cause a system to violate timing and resource requirements using system models. Our work is different from existing work in automatic test generation for resource consumption because it can provide an efficient and effective test selection method in the presence of seeded energy-related faults; this aspect has received little attention in the literature.

VII. CONCLUSIONS

In this paper, we have outlined a method for testing energy consumption in embedded systems. The method makes use of energy requirements as expressed in EAST-ADL architectural models, transforms these requirements together with the components interfaces to priced timed automata, and uses statistical model checking in order to identify relevant test cases. We use random simulations to create test suites containing input parameters and energy signals. Given the large number of

potential test suites, we select several simulations showing the nominal and the worst-case energy consumption using statistical model checking. A pilot case study of the method, using a Brake-by-Wire system provided by Volvo Group Trucks Technology, Sweden, indicates that statistical model checking is suitable for generating test suites for energy consumption. The evaluation shows that the test suite generation method is efficient in terms of time required to generate tests. We have proposed in this study to evaluate the fault detection ability of these test suites by seeding energy consumption modeling errors, resulting by changing the rate of cost consumption over time. Our results suggest that an approach that selects test suites showing diverse energy consumption can increase the fault detection ability.

Future work aims at extending our method to generating tests for other types of resources too, and apply it more extensively on actual industrial cases to expose its strengths as well as limitations.

ACKNOWLEDGMENTS

This research was supported by the Swedish Research Council (VR) through the "*Adequacy-based testing of extra-functional properties of embedded systems*" project. Under the EU project AAL–2014–1–08, this work is also supported by a grant of the Swedish Governmental Agency for Innovation System (Vinnova) as well as by the Swedish Knowledge Foundation (KKS), within the DPAC (Dependable Platforms for Autonomous Systems and Control) research profile.

REFERENCES

[1] The AUTomotive Open System ARchitecture (AUTOSAR) Standard. Available from http://www.autosar.org/, 2014.

[2] Aynur Abdurazik, Zhenyi Jin, Liz White, and Jeff Offutt. Analyzing software architecture descriptions to generate system-level tests. In *Workshop on Evaluating Software Architectural Solutions*, 2000.

[3] Shaukat Ali, Lionel C Briand, and Hadi Hemmati. Modeling robustness behavior using aspect-oriented modeling to support robustness testing of industrial systems. *Software & Systems Modeling*, 11(4):633–670, 2012.

[4] Paul Ammann and Jeff Offutt. *Introduction to Software Testing*. Cambridge University Press, 2008.

[5] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *Computer*, 40(12), 2007.

[6] Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Larsen, Paul Pettersson, Judi Romijn, and Frits Vaandrager. Minimum-cost reachability for priced time automata. In *Hybrid Systems: Computation and Control*, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2001.

[7] Antonia Bertolino and Paola Inverardi. Architecture-based software testing. In *International Software Architecture Workshop*, pages 62–64. ACM, 1996.

[8] Antonia Bertolino, Paola Inverardi, and Henry Muccini. Software architecture-based analysis and testing: a look into achievements and future challenges. *Computing*, 95(8):633–648, 2013.

[9] Hans Blom, Henrik Lönn, Frank Hagl, Yiannis Papadopoulos, Mark-Oliver Reiser, Carl-Johan Sjöstedt, De-Jiu Chen, Fulvio Tagliabò, Sandra Torchiaro, and Sara Tucci. EAST-ADL: An Architecture Description Language for Automotive Software-Intensive Systems. *EAST-ADL White Paper*, 1, 2013.

[10] Peter Bulychev, Alexandre David, Kim G Larsen, Axel Legay, Guangyuan Li, and Danny Bøgsted Poulsen. Rewrite-Based Statistical Model Checking of WMTL. In *Runtime Verification*. Springer, 2013.

[11] Peter Bulychev, Alexandre David, Kim Gulstrand Larsen, Marius Mikučionis, Danny Bøgsted Poulsen, Axel Legay, and Zheng Wang. UPPAAL-SMC: Statistical Model Checking for Priced Timed Automata. *Workshop on Quantitative Aspects of Programming Languages and Systems*, 2012.

[12] Giovanni Denaro, Andrea Polini, and Wolfgang Emmerich. Early performance testing of distributed software applications. In *ACM SIGSOFT Software Engineering Notes*, volume 29, pages 94–103. ACM, 2004.

[13] Vahid Garousi, Lionel C Briand, and Yvan Labiche. A uml-based quantitative framework for early prediction of resource usage and load in distributed real-time systems. *Software & Systems Modeling*, 8(2):275–302, 2009.

[14] Mark Grechanik, Chen Fu, and Qing Xie. Automatically finding performance problems with feedback-directed learning software testing. In *International Conference on Software Engineering (ICSE)*, pages 156–166. IEEE, 2012.

[15] Jonathan Hammond, Rosamund Rawlings, and Anthony Hall. Will it work?[requirements engineering]. In *International Symposium on Requirements Engineering*, pages 102–109. IEEE, 2001.

[16] Mary Jean Harrold. Architecture-based regression testing of evolving systems. In *International Workshop on the Role of Software Architecture in Testing and Analysis*, 1998.

[17] Zhen Ming Jiang and Ahmed E Hassan. A survey on load testing of large-scale software systems. *IEEE Transactions on Software Engineering*, 41(11):1091–1118, 2015.

[18] Zhen Ming Jiang, Ahmed E Hassan, Gilbert Hamann, and Parminder Flora. Automated performance analysis of load tests. In *International Conference on Software Maintenance*, pages 125–134. IEEE, 2009.

[19] Zhenyi Jin and Jeff Offutt. Deriving tests from software architectures. In *International Symposium on Software Reliability Engineering*, pages 308–313. IEEE, 2001.

[20] Eun-Young Kang, Eduard Paul Enoiu, Raluca Marinescu, Cristina Seceleanu, Pierre-Yves Schobbens, and Paul Pettersson. A Methodology for Formal Analysis and Verification of EAST-ADL Models. *Reliability Engineering and System Safety*, 120, 2013.

[21] Ivano Malavolta, Patricia Lago, Henry Muccini, Patrizio Pelliccione, and Antony Tang. What industry needs from architectural languages: A survey. *IEEE Transactions on Software Engineering*, 39(6):869–891, 2013.

[22] Raluca Marinescu, Eduard Paul Enoiu, and Cristna Seceleanu. Statistical analysis of resource usage of embedded systems modeled in east-adl. In *Annual Symposium on VLSI*, pages 380–385. IEEE, 2015.

[23] Clementine Nebut, Franck Fleurey, Yves Le Traon, and J-M Jezequel. Automatic test generation: A use case driven approach. *IEEE Transactions on Software Engineering*, 32(3):140–155, 2006.

[24] Alexander Pretschner, Manfred Broy, Ingolf H Kruger, and Thomas Stauner. Software engineering for automotive systems: A roadmap. In *2007 Future of Software Engineering*, pages 55–71. IEEE Computer Society, 2007.

[25] Debra J Richardson and Alexander L Wolf. Software testing at the architectural level. In *International Software Architecture Workshop and International Workshop on Multiple Perspectives in Software Development*, pages 68–71. ACM, 1996.

[26] Mary Shaw and David Garlan. *Software architecture: perspectives on an emerging discipline*, volume 1. Prentice Hall Englewood Cliffs, 1996.

[27] Elaine J Weyuker and Filippos I Vokolos. Experience with performance testing of software systems: issues, an approach, and case study. *IEEE transactions on software engineering*, 26(12):1147, 2000.

[28] Tao Yue and Shaukat Ali. 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*, pages 133–145. Springer, 2012.

[29] Jian Zhang and Shing Chi Cheung. Automated test case generation for the stress testing of multimedia systems. *Software: Practice and Experience*, 32(15):1411–1435, 2002.

[30] Jian Zhang, Shing-Chi Cheung, and Samuel T Chanson. Stress testing of distributed multimedia software systems. In *Formal Methods for Protocol Engineering and Distributed Systems*, pages 119–133. Springer, 1999.