Model-based Mutation Operators for Timed Systems

Ortiz Vega, James Jerson; Perrouin, Gilles; Amrani, Moussa; Schobbens, Pierre-Yves

DOI:
10.1109/QRS.2018.00045

Publication date:
2018

Document Version
Peer reviewed version

Model-based Mutation Operators for Timed Systems: A Taxonomy and Research Agenda

James Jerson Ortiz Vega, Gilles Perrouin, Moussa Amrani, Pierre-Yves Schobbens
PReCISE, Namur Digital Institute
Faculty of Computer Science, University of Namur, Belgium
Email: {James.OrtizVega, Gilles.Perrouin, Moussa.Amrani, Pierre-Yves.Schobbens}@unamur.be

***Abstract*—Mutation testing relies on the principle of artificially injecting faults in systems to create mutants, in order to either assess the sensitivity of existing test suites, or generate test cases that are able to find real faults. Mutation testing has been employed in a variety of application areas and at various levels of abstraction (code and models). In this paper, we focus on model-based mutation testing for timed systems. In order to cartography the field, we provide a taxonomy of mutation operators and discuss their usages on various formalisms, such as timed automata or synchronous languages. We also delineate a research agenda for the field addressing mutation costs, the impact of delays in operators specification and mutation equivalence.**

***Index Terms*—Model-Based Testing; Mutation Testing; Timed Automata; Real-Time Systems; Mutation Operators Taxonomy.**

I. Introduction

Timed Systems are currently deployed into many safety-critical and/or embedded applications, e.g. airplanes, satellites, trains, automotive and nuclear systems. Such systems are usually composed of deeply integrated hardware and software components developed under severe resource limitations and high quality requirements. When a failure occurs in such a system, it may result in enormous costs, human injuries and life losses. Consequently, a systematic verification & validation approach for timed systems is a crucial issue for those systems whose correctness not only depends on the correct functioning, but also on meeting time constraints.

Formal verification techniques aim at exhaustively checking the correct behaviour of timed systems against their requirements [1]. Though, those techniques suffer from scalability issues and mathematical sophistication that directly hamper their large-scale usage. Testing remains the *de facto* approach, especially when timed systems under consideration are *cyber-physical*: such systems are composed of interacting software and hardware components. Model-Based Testing [2], [3] consists in producing test specifications entirely or in part from both the system requirements and a model describing selected (non-)functional aspects of the system under test. The use of models helps managing timed systems' complexity by working with abstractions focusing on the appropriate properties to be validated.

Mutation Testing (MT) [4] is an established technique for two usages: (i) evaluating the effectiveness of the test suites [5], [6], [7]; and (ii) supporting test generation [8], [9], [6]. It works by injecting artificial defects, called *mutations*, into the code or the model under test, yielding *mutants*. Test suites may be evaluated against mutants by comparing outputs of the original system against those of the mutant: when different, the mutant is said to be *killed* (or *distinguished*) by test cases. The effectiveness of test suites is then measured by the *mutation score*, i.e. the ratio between killed mutants over the total number of mutants. To obtain an effective test suite, we expect to kill all the mutants. However, some mutants may be *equivalent*, i.e. they exhibit the exact same behaviour as the original system although syntactically mutated, and therefore cannot be killed. As a consequence, appropriately detecting equivalent mutants (known as the Equivalent Mutant Problem within MT [10]) is an important challenge to improve the overall testing framework efficiency.

MT can also be leveraged to generate specific test cases for killing mutants, either at the *code level*, by exploiting dynamic symbolic execution to generate test data for mutated paths [8], or at the *model level*, by using specific techniques such as input-output conformance (or ioco) [11] or model-checking of a violated temporal property [12].

Model-Based Mutation Testing (MBMT) therefore combines the strengths of both Model-Based Testing and Mutation Testing, and has been largely used as a complementary technique to code-based mutation technique. Aichernig *et al.*[13] report that model mutants lead to tests that are able to reveal implementation faults that were neither found by manual tests, nor by the actual operation of an industrial system. In addition, model-based mutation's premise is to identify defects related to missing functionality and misinterpreted specifications [14]. This is desirable since code-based testing fails to identify these kinds of defects [15], [16].

This paper aims at evaluating the relevance of Mutation Testing for timed systems. We propose to survey the existing literature contributions to retrieve the mutation operators used for a large panel of timed systems, and organise them in a comprehensive manner, for future evaluation, in a simple two-level taxonomy. More specifically, this paper provides:

- A survey on Mutation Testing for Timed Systems;
- A two-level taxonomy of mutation operators spanning the whole spectrum of Timed Systems;
- A provisional research agenda, covering notably the need to further validate time-related operators and mutation equivalence.

Author	Title	Published	Ref.
Aichernig *et al.*	Time for Mutants - Model-Based Mutation Testing with Timed Automata	Tests& Proofs'13	[11]
Nilsson *et al.*	Mutation-Based Testing Criteria for Timeliness	COMPSAC'04	[17]
Hanh & Binh	Mutation Operators for Simulink Models	KSE'12	[18]
Du Bousquet & Delaunay	Towards Mutation Analysis for Lustre Programs	ENTCS'08	[19]

TABLE I
SELECTED CONTRIBUTIONS FOR MODEL-BASED MUTATION OF TIMED SYSTEMS

The remainder of this paper is organised as follows. Section II briefly presents our selection protocol for retrieving the literature contributions on mutation operators for timed systems; threats to validity are discussed later in Section VII. Section III discusses the general principles that governs the construction of our taxonomy. Section IV and V discusses operators that works with explicit- (e.g. Timed Automata) and implicit-time models (mainly, MathLab SimuLink and Esterel Scade), respectively. Section VI describes our research agenda for the field, and Section VIII wraps up with concluding remarks and future work.

II. Survey on Timed Mutation Operators

To collect mutation operators, we surveyed the online scientific literature in popular search engines (SpringerLink, ScienceDirect, Google Scholar, Scopus, IEEE XPlore and ACM Digital Library). We completed our initial set of contibutions by manually investigating papers based on the authors' knowledge. We also operated a backward snowballing of the bibliographical references [20].

We used an internal Github repository to collect the resulting contributions: the initial set was composed of 61 studies, from which we explicitly excluded the ones that did not describe mutation operators explicitly, or did not take time-related mutations into account. For example, Aichernig *et al.* [21] mutated hybrid systems by abstracting away both physical, continuous aspects and discretising time, resulting in non-timed labelled transitions systems. Furthermore, when different contributions specified the same (sets of) operators, we selected the most precise one. Our goal was not to focus on the chronological developments of these operators, but rather to classify them and evaluate their similarities. This additional filtering step resulted in six distinct contributions, published between 2004 and 2015, that formed the base material for our taxonomy presented in the next section. As a matter of fact, all the recent contributions (e.g., [22]) tend to reuse existing operators rather than providing new ones. The studies we retained for our taxonomy are presented in Table I.

III. Taxonomy Principles

Building a taxonomy of mutation operators was hindered by the fact that many of the contributions we retrieved study the matter in silos: they often position themselves only with closely related work, in some cases only focusing on the same formalism. To build a general taxonomy of mutation operators, it becomes necessary to place the notion of time and formalisms in a broader perspective. Furia *et al.* [23] proposed an interesting classification on how time is represented and manipulated in different programming paradigms and languages: when abstracted for being represented in computers, time is interpreted over a specific *domain*, and manipulated through a *language* that possesses specific *features*. These are the characteristics that determine time-dependent reasoning.

The *semantic domain* is the mathematical numerical set whose intrinsic mathematical properties influences both the manipulation of time and the scope of properties available. However, although most of real-time systems are based on real time values, these values are often abstracted away or discretised to simplify the programming. Since mutation operators are pure syntactic changes in the real-time system representation, this should represent a better classifier than the mathematical domain for time. Among all formalisms' features described by Furia *et al.*, we identified two that are largely represented in the contributions we retrieved. First, the *property type* indicates how relevant events may occur in the system. A *quantitative* constraint specifies a precise duration (e.g., a car break should activate within 50 ms after the pedal break is pushed); whereas a *qualitative* constraint only imposes a (partial) order between them (e.g., the break should occur only some time after). Second, the *synchronicity* of the formalism is important: *synchronous* systems impose changes in modules to occur at the same time, or at times rigidly related; whereas *asynchronous* systems allow independent progress of each module.

Although not a general observation, we noticed that both features are often related: asynchronous systems make use of quantitative constraints; while synchronous systems relate events/signals qualitatively. As a matter of fact, the way time elapses in synchronous systems is beyond the language's syntactic domain: it belongs to the semantic domain, where computations occur in cycles that may be aligned to precise timing. These observations form the basis, as well as the top-level criterion of our taxonomy: for asynchronous systems with quantitative constraints, the notion of clock is *explicitly* part of the underlying formalism; while for synchronous systems with qualitative constraints, the notion of clock is only *implicit*. Since the contributions we retrieved are largely clustered according to specific formalism, we simply distinguish further those formalisms.

Table II shows the mutation operators retrieved from the considered contributions, sorted by formalisms in implicit- and explicit-clock models.

	Formalisms	Mutation Operators
Explicit clock models	**Timed Automata**	- Change action [11] - Change target [11] - Change source [11] - Change guard [11] - Negate guard [11] - Change invariant [11] - Sink location [11] - Invert reset [11]
	Timed Automata+	- Execution time [17] - Inter-arrival time [17] - Pattern offset [17] - Lock time [17] - Unlock time [17] - Hold time shift [17] - Precedence constraints [17]
Implicit clock models	**Simulink**	- Types replacement [18] - Variable change [18] - Variable negation [18] - Statement change [18] - Delay change [18] - Relational replacement [18] - Logical replacement [18] - Aritmetic replacement [18]
	Lustre/Scade	- Variable change [19] - Arithmetic replacement [19] - Logical replacement [19] - Relational replacement [19] - Unary insertion [19] - Unary replacement [19] - Temporal replacement [19] - Subprogram replacement [19]

TABLE II
TAXONOMY OF THE MUTATION OPERATORS OF TIMED SYSTEMS

IV. EXPLICIT CLOCK MODELS

Timed Automata (TA) [24] are amongst the most studied formalisms with explicit manipulation of time. TA are an extension of Finite State Automata with a set of real-valued clocks increasing at the same rate. Each transition has a (possibly empty, i.e. `true`) guard, an action and a (possibly empty) clock reset set. A guard defines a condition enabling state change, and consists of an expression composed of a conjunction of clock comparisons (i.e. of the form $x \sim c$ where c is an integer, and $\sim$ is one of the symbols $\{<, \leq, =, \geq, >\}$). Clock resets allow to measure time elapse between states, which may contain invariants, also expressed with the same kind of expressions. Figure 1 depicts an example of TA with two states S_1 and S_2 and two transitions. The invariant $x < 7$ is defined over S_1, while the transition from S_1 to S_2 operates on action a, is guarded by $x > 3$ and resets the clock x.

Several extensions of TA have been also considered for mutation analysis: TA with Inputs/Outputs (TAIO) [25] partition the actions into two disjoint sets for inputs and outputs; and TA have been combined with a task model (TAT) to capture scheduling and execution of tasks [26]. TAIO and TAT appear in the *Timed Automata+* category.

A. Timed automata operators

Aichernig *et al.* [11] provided eight operators to mutate TA defined in the Uppaal specification format. Four of them are not time-related: *change Action*, *change source/target* and *sink location*. The first time-related operator, *change guard* alters the inequality within the guard constraint (e.g., replacing $x > 3$ by $x \geq 3$). The *negate guard* operator replaces a transition boolean guard by its logical negation. Another operator applicable to transitions, *invert reset*, selects one clock variable and either adds it to the list of clocks to be reset during the transition if it is absent, or removes it from the list if it is present. Finally, *change invariant* adds one time unit (to mimic off-by-one errors) to the invariant constraint (which represents an inequality between a clock value and a positive real value) in an automaton state (e.g., $x < 8$ in $S1$). While guard mutation operators can be thought of as timed modifications of usual MBMT operators [27], *change invariant* and *invert reset* are specific to timed automata, as mentioned by the authors [11]. In addition to providing mutation operators, the authors designed a test generation framework based on the input-output extension of the ioco conformance relation [28]. Tests are created from a mutant only if the mutant violates the conformance relation. The authors applied their framework on a car alarm system and found that the operators working on automata structure (e.g., *change target*) yielded the highest mutation scores (94.7%). In contrast, time-related operators yielded mutation scores lower or equal to 60%. This is an indication that time-related mutants are hard to kill.

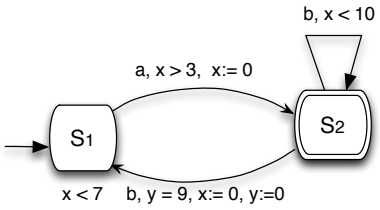

Fig. 1. A timed automaton with two clocks x and y.

B. Extended Time Automata Operators

Nilsson *et al.* [17] first extend the TA formalism with a task model, as they focus on their *timeliness*, i.e. the ability for a system to meet its deadlines. Tasks refer to locations of the underlying TA and are organised in an execution queue. This queue specifies for each task the time remaining for the task; a set of system-wide resources (semaphores) and the interval of time they can be locked by the task; as well as precedence relations with other tasks. Generally, the proposed operators affect time by shifting it with a certain amount Δ. Nilsson *et al.* define specific task set operators: *execution time* affects a task's execution time; *hold time shift* and *lock/unlock time* operators either shift the whole lock/unlock time interval for a resource, or only one of its bounds; and *precedence constraints* operators change precedence relations between pairs of tasks. The authors also define automata operators that affect both invariant and guard constraints either for a given location (*inter-arrival time*), or for the initial location (*pattern offset*).

While the initial paper only illustrates the operators, a more complete evaluation on a robot arm was conducted in [29]. The authors of this contribution found that randomly generated test suites were unable to find any time-related fault. More interestingly, the same was observed for manually created test suites. This experiment is an indication that timing faults are subtle and that manually creating effective test cases to find timing faults is difficult.

V. Implicit Clock Models

A number of formalisms have an *implicit* notion of clocks, especially when they follow the synchronous hypothesis [30]. This hypothesis relies on a logical, abstract discrete time corresponding to cyclic execution instants: some inputs occur into the system, triggering internal computations that propagate data according to the formalisms' control definitions, until output data are computed for defining a new, global state. This abstract logical time does not directly corresponds to the actual physical time; all that matters is that all computations converge and finish before the next instant occurs.

We retain for our study the two main formalisms and tools widespreadly used to model embedded critical systems: MathLab SimuLink and Esterel SCADE. Both rely on graphical representations that define a model as a collection of subsystems consisting of functional blocks that are connected through links/wires that transport data: the output of one block is transmitted to the input of another one, each block being defined from the tool's library or basic blocks, a user-defined combination of them, or even a full, hierarchical subsystem. Figure 2 shows two simple examples of such system definitions. Both SimuLink and SCADE have textual representations that could be used for code-level mutations; however, the correspondence between the graphical and textual representations is not always guaranteed. While SCADE has a well-defined semantics in terms of Lustre semantics, SimuLink suffers from the lack of formal semantics, which results in possible discrepancies between code generated by different tools from the same SimuLink models. As a consequence, mutations operated on textual representations of SimuLink models may not straightforwardly trace back to their original model.

A. MathLab SimuLink

Hanh *et al.* [18] provided one of the most compelling mutation operators collection for SimuLink, aimed at validating a test suite. Mutation operators are spread into five classes: *type* mutations (TRO) switch variable/constant types with another, compatible one; *variable* mutations (VCO/VNO) either change a variable's value (by adding, multiplying or setting the value), or negating it (for numerical and boolean values); *constant* mutations (CCO/CRO/DCO) either change constants' values (by incrementing, decrementing or resetting them) or replacing them by a predefined value, or operate on the *Delay* SimuLink operator to change its value; *statement* mutations (SCO/SSO) operate on the *Switch* operator by either changing its threshold value, or swapping the possible results; and *expression* mutations (RORO/AORO/ASRO/LORO) operate by replacing arithmetico-boolean operators with another compatible one (e.g. a '+' with a '-'). The approach is validated on a small SimuLink example (a quadratic model). The authors have improved their methodology by using Genetic Algorithms to reduce the number of produced mutants [31] and automatically generate test data [32], but both contributions are based on the same set of mutation operators. Similarly, Yongfeng *et al.* [33] rely on the same operators but combine them in different ways to assess which combinations provide the best results; however, how they validate their approach is by far not clear in the paper.

Stephan and his colleagues [34], [35] used mutation operators for detecting three types of SimuLink models clones: exact clones are models that differ only in their layout properties (e.g., location or background colour of blocks); renamed clones are models that are topologically identical up to the blocks, wires, etc. names; and near-miss clones are models that differ in particular structural aspects. From a behavioural perspective, the mutation operators for the two former clone types are irrelevant; we therefore focus on the four operators belonging to the latter: mADBD and mADBS adds/deletes block as destination and source respectively, i.e. introduces or removes a block as an output/input of a hierarchical block; mCBT changes a block's type without changing its name, while respecting its input/output links constraints; and mCSCH changes a subsystem's clone hierarchy, i.e. factor out a portion of a design into an independent block. The contributions are validated on the classical Automotive Power Window case study shipped with SimuLink and an open-source model describing an Advanced Vehicle Simulator.

Zhan and Clark [36] used three types of mutation operators to assess the quality of a test suite: add/multiply/assign modify the value carried on a block input by/with a certain value that is parameterisable by the user and that can be applied in various locations. This simulates signal perturbations that model initialisation, assignments, condition checks or functions/subsystems faults. They automatically produce the corresponding mutants while preserving the model's correctness. The approach is validated using two small models extracted from SimuLink library and an average-size model extracted from industrial cases.

N. He and her colleagues [37], [38] targeted test case generation for SimuLink models. In [37], although the mutations seem to alter the graphical representation, SimuLink models are actually translated into C, and it is not clear at which level mutations are actually performed. The list of mutation operators is given in [38]: RC alters a constant by incrementing, decrementing or resetting its value; ABS inserts an absolute value to numerical inputs; UOI inserts negation operators (for boolean or arithmetic inputs); INC adds a constant value to an input; RR and RL swap arithmetic and boolean operators respectively. Their studies are validated using a combination of library models from SimuLink and industrial fragments from a case study of a steering anti-catchup from Ford.

Fig. 2. Implicit-Clock model example: a SimuLink model on the left and a SCADE model on the right.

Araujo *et al.* [39] translated the guidewords of the Hazard and Operability studies (HAZOP) into mutation operators and provided a classified definition very similar to [18]: the classification admits the same categories (type; variable; constant; block, which corresponds to *statement*; and expression) that have small variations: they propose a variable replacement operator (VRO) that swaps the connections between compatible variables as inputs of a block, and a subsystem change operator (SCO) that is similar, but acts on a higher granularity; the block removal operator (BRO) which removes blocks from a model. Their framework is validated on a small case study representing an electronic temperature regulator.

Runge [40] proposed a slightly different classification over general mutation operators extracted from [4] and SimuLink-specific ones borrowed from [39], [18], [32], [38]: operators are classified according to the model entities they alter, i.e. at the Block/Component level; on the Control Flow; or on the Data Flow. The mutation framework was implemented in a tool and evaluated on industrial cases.

B. Esterel Scade

Delaunay and his colleagues [41], [42], [19] study the possibility of defining mutation operators on Lustre, the academic, textual version of Esterel SCADE, in order to assess the validity of test suites. The mutation operators they use are directly inspired from operators defined for the Fortran language: the authors recognise that some of them are simply not directly applicable to Lustre (e.g. mutation operators for array typically introduce mutants that do not compile properly), and that more extensive research is needed to properly capture Lustre-specific operators, especially for the time-manipulation syntactic constructs (`pre`, `followed-by`, `current` and `when`).

Papailiopoulou [43] proposed a set of six mutation operators (classes) that are for the most of them very classical (e.g., switching relational and arithmetico-boolean operators), but introduced two interesting variations: she proposed to switch the `pre` operator with the `not`, as well as the `and`/`or` operators with the `followed-by`; and to switch library temporal operators (reflecting the classical temporal constructs of Temporal Logics, e.g. always/globally) among each others.

VI. Research Directions

We recalled in the introduction the main challenges MBMT faces, and discussed the specificities of Timed Systems in Section III. In this section, we revisit those challenges in light of our taxonomy and draw potential research directions.

A. Mutation Testing Framework

1) Overview: Costs of generating mutants and running tests on them have long been seen as major impediments to the deployment of mutation testing in practice [10], [4]. For explicit clock models, challenges arise both from building the testing infrastructure that sometimes needs to emulate the actual system [11] or performing tests [29]. For implicit clock models, this aspect is almost never discussed since time is controlled externally. Nevertheless the analysis can be costly in terms of computational resources [38].

2) Research Directions: Model-in-the-Loop approaches, where the model is directly integrated in the timed systems could be considered as helpful to support full automation of mutation testing and reduce the simulation effort. However, it may be dangerous to evaluate mutants on cyber-physical systems (robot arms, etc.) due to the safety risks mutant execution yield. As an ever increasing part of systems can be generated and simulated (Simulink and Scade environment target full code generation in some cases) the cost of building simulation infrastructure should decrease. An additional improvement would be to adopt mutant schemata techniques to speed up analysis [44], [45].

B. On Time-specific Mutation Operators

1) Overview: Mutation operators explicitly manipulating time elements have a different nature depending on the clock model. For explicit clock models, they simply reflect algebraic manipulations targeted at clock elements: for TA, they treat clock variables as normal variables, e.g. negated inside an expression, or inverted inside the reset set [17]; while for TA+, they are simply treated as arithmetic expressions that can be altered (typically with an offset or reset) [29]. For implicit models, since no direct access to time is explicit, operators are applied on specific blocks (e.g., *Delay* or *Switch* SimuLink blocks [18]; `pre`, `followed-by` and the like for Scade/Lustre [19]). Since these mutation operators directly alter the timeliness execution of the corresponding models, they introduce subtle errors that are difficult to detect without specific test cases. Nilsson and Offut [29] compared test cases generated from mutation operators with randomly generated ones, and observed a clear superiority of the former ones. Similarly, Hanh *et al.* [18] and Araujo *et al.* [39] noticed that mutation operators on specific, time-related SimuLink blocks

both produce fewer mutants that are also difficult to kill (notice that the contributions on Scade have not evaluated the effect of the time-specific operators explicitly).

2) Research Directions: As a consequence, using time-specific mutant operators for explicit clock models seem to reveal subtle errors and should be investigated further (specifically on TA); whereas operators for implicit clock models seem promising, but require further investigation for both formalisms, because they are not fully exploited and no study evaluated objectively their impact on test case coverage. Such studies need to be conducted to further assess the relevance of mutation testing for timed systems. Especially because timed systems are often critical ones, (e.g., embedded systems) strong empirical evidence must be given in order for mutation testing to be considered as reliable test generation technique.

C. On Mutation Equivalence

1) Overview: The Equivalent Mutant Problem EMP is a well-known issue in mutation analysis [10], [46], [47]. This is particularly problematic with respect to both generation and assessment of test suites: in the former case, resources are spent on trying to kill non-killable mutants; while in the later case, resources are spent on skewing the mutation score (a 100% mutation score is impossible to reach when there are equivalent mutants). At the model level, the EMP can be expressed in terms of language equivalence between (non-timed) automata, for which exact solutions and simulations exist [48]. For explicit clock models, timed bisimulation [49] and timed trace equivalence [50] techniques can be used for reasoning about behavioural equivalence between TA. However, while the timed bisimulation problem is decidable in EXPTIME [49], trace equivalence problems are undecidable [50]. However, timed bisimulation has not been widely used to detect equivalent mutants, but this may be due to the so-called state explosion problem [50] caused by both discrete and timing properties. Siavashi *et al.* [51] use mutation operators for TA to generate mutants and timed bisimulation for detecting and eliminating equivalent mutants. In particular, their approach is based on the verification of reachability and deadlock-freeness properties in the Uppaal tool [52]. The *change invariant* and *invert reset* operators systematically yielded equivalent mutants: this shows the limits of the equivalence relation offered for time-related mutants. Aichernig *et al.* use mutation for test generation and did not report whether equivalent mutant were produced by their operators [11]. The same can be observed for extended timed automata [17], [29].

For implicit clock models, Hanh *et al.* adopted a different notion of equivalence (named "test-equivalence") that is adding new tests until a mutant is killed or a certain limit is reached (in this case, mutants are only "probably equivalent" [48]). It is not obvious how this equivalence detection approach was automated. He *et al.* [38] offered to use Formal Concept Analysis to create concept lattices to group mutants that are likely to be killed by the same test cases. The authors provided an approximated method to identify mutant clusters but unfortunately do not report on its efficiency regarding equivalence in the experimentation section. Du Bousquet *et al.* used a model-checker to assess whether Lustre mutants satisfy the same properties as the original system [42], [41]. The authors did not detail the equivalent mutants found and which operators generated them.

2) Research Directions: While there are promising attempts to address the EMP for timed systems, we are not there yet. The general problem is undecidable but there is room for approximate techniques (such as simulations or bisimulations) and operator-specific equivalent detection strategies. Approximate computing was recently used at the code level to speed up mutation analysis [53], [54]: we think such techniques might be adapted to the EMP for timed systems as well. An interesting track to explore is to use timed reduction techniques with reachability algorithm and decision algorithms [55].

VII. Threats to Validity

The main threat to validity in this research is the way we elicited the source contributions on which our taxonomy is based. We did not adopt a Systematic Literature Review approach [56] aiming at a repeatable search protocol, but rather targeted a lightweight protocol because mutation operators are barely a first-class keyword in contributions' titles. Instead, they are part of a larger test assessment or test generation framework, and are rarely discussed as such (some contributions mention mutation operators without explicitly listing them). Similarly, we did not perform a rigourous snowballing as described by Wohlin [20]: we only performed a backward snowballing starting from all the papers cited in this survey, and marginally operated forward snowballing on the main papers listed in Table I, using the same exclusion criteria as the ones used for the search protocol. Snowballing may have the silo effect of focusing all contributions retrieved to the same formalisms. As a general observation however, all recent contributions do not introduce newer, or different operators, that the ones we listed here.

The taxonomy proposed in this paper operates at two levels: first, the way clocks are manipulated; and second, the formalism itself. We showed that mutation operators work with the formalism's topology as well as the expressions available for the programmer, and that explicit-clock formalisms usually manipulate clock-specific mutation operators similarly than other expressions (e.g., a clock variable is simply a variable, submitted to the same mutations than others). While we believe these principles sound enough for capturing the mutation operators classes for both explicit and implicit clock formalisms, hybrid systems may represent a threat to the validity of our classification, since operators manipulating continuous variables (with derivatives or integration, among others) may differ. However, these kinds of operators are already integrated in implicit-clock formalisms (SimuLink and Scade define them as library blocks), and can therefore be manipulated with topological mutation operators.

VIII. Conclusion

In this paper, we provided a taxonomy of mutation operators for timed systems. Our goal was to understand the main categories of time-related formalisms where model-based mutation testing have been applied and how time affects mutation. We found that, if there is a variety of operators proposed, their evaluation and comparison in the large is still missing. This may be explained by the lack of freely available models or proprietary tools which are *de facto* standards for timed systems. However, evaluations offered so far demonstrate that time-related operators are mimicking subtle time faults that are hard to find with manually created or randomly generated test suites. This therefore motivates further research in this area.

Notably, we think that the equivalent mutant problem is still insufficiently addressed, as time-specific detection strategies are often missing. This forms the first item of our future work. The second one concerns the creation of benchmarks for time-related model-based mutation testing, in order to provide strong empirical evidence that mutation testing is relevant as a (Model-Based) Testing approach for embedded and critical contexts. In particular, it would be interesting to evaluate empirically whether Mutation-Based Testing may discover subtler errors than traditional Modified Condition / Decision Coverage (MC/DC), considered as a standard for certification purposes.

Acknowledgment

Gilles Perrouin is a Research Associate at the FNRS (Fonds National de la Recherche Scientifique). Moussa Amrani was supported by the Walloon Region SkyWin Project D-DAMS Number 7513.

References

[1] F. Wang, "Formal verification of timed systems: a survey and perspective," *Proceedings of the IEEE*, vol. 92, no. 8, pp. 1283–1305, 2004.

[2] J. Tretmans, *Formal Methods and Testing – An Outcome of the FORTEST Network (Revised Papers Selection)*. Springer-Verlag, 2008, ch. Model Based Testing with Labelled Transition Systems, pp. 1–38. [Online]. Available: http://dl.acm.org/citation.cfm?id=1806209.1806210

[3] J. Zander, I. Schieferdecker, and P. J. Mosterman, *Model-Based Testing for Embedded Systems*. CRC Press, 2017.

[4] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Trans. Softw. Eng.*, vol. 37, no. 5, pp. 649–678, Sep. 2011. [Online]. Available: http://dx.doi.org/10.1109/TSE.2010.62

[5] J. H. Andrews, L. C. Briand, Y. Labiche, and A. S. Namin, "Using Mutation Analysis for Assessing and Comparing Testing Coverage Criteria," *Software Engineering, IEEE Transactions on*, vol. 32, no. 8, pp. 608–624, 2006.

[6] J. Offutt, "A mutation carol: Past, present and future," *Information and Software Technology*, vol. 53, no. 10, pp. 1098–1107, Oct. 2011.

[7] M. Gligoric, A. Groce, C. Zhang, R. Sharma, M. A. Alipour, and D. Marinov, "Comparing non-adequate test suites using coverage criteria," in *ISSTA*. ACM, 2013, pp. 302–313.

[8] M. Papadakis and N. Malevris, "Automatic mutation test case generation via dynamic symbolic execution," in *ISSRE*. IEEE, 2010, pp. 121–130.

[9] G. Fraser and A. Arcuri, "Achieving scalable mutation-based generation of whole test suites," *Empirical Software Engineering*, pp. 1–30, 2014.

[10] M. Papadakis, M. Kintis, J. Zhang, Y. Jia, Y. L. Traon, and M. Harman, "Mutation testing advances: An analysis and survey," *Advances in Computers*, vol. 112, 2018.

[11] B. K. Aichernig, F. Lorber, and D. Nickovic, "Time for mutants - model-based mutation testing with timed automata," in *Tests and Proofs - 7th International Conference, TAP 2013, Budapest, Hungary, June 16-20, 2013. Proceedings*, ser. Lecture Notes in Computer Science, M. Veanes and L. Viganò, Eds., vol. 7942. Springer, 2013, pp. 20–38.

[12] G. Fraser and F. Wotawa, "Using model-checkers for mutation-based test-case generation, coverage analysis and specification analysis," in *Software Engineering Advances, International Conference on*, Oct 2006, pp. 16–16.

[13] B. K. Aichernig, J. Auer, E. Jöbstl, R. Korosec, W. Krenn, R. Schlick, and B. V. Schmidt, "Model-based mutation testing of an industrial measurement device," in *Tests and Proofs*, ser. LNCS, vol. 8570. Springer, 2014, pp. 1–19.

[14] T. A. Budd and A. S. Gopal, "Program testing by specification mutation," *Computer Languages*, vol. 10, no. 1, pp. 63–73, Jan. 1985.

[15] W. E. Howden, "Reliability of the path analysis testing strategy." *IEEE Transactions on Software Engineering*, vol. 2, no. 3, pp. 208–215, 1976.

[16] J. M. Voas and G. McGraw, *Software Fault Injection: Inoculating Programs Against Errors*. John Wiley & Sons, Inc., 1997.

[17] R. Nilsson, J. Offutt, and S. F. Andler, "Mutation-based testing criteria for timeliness," in *Proceedings of the 28th Annual International Computer Software and Applications Conference - Volume 01*, ser. COMPSAC '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 306–311. [Online]. Available: http://dl.acm.org/citation.cfm?id=1025117.1025515

[18] L. T. M. Hanh and N. T. Binh, "Mutation operators for simulink models," in *Knowledge and Systems Engineering (KSE)*, 2012.

[19] L. Du Bousquet and M. Delaunay, "Towards mutation analysis for lustre programs," *Electronic Notes in Theoretical Computer Science*, vol. 203, no. 4, pp. 35–48, 2008.

[20] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, 2014, pp. 38:1–38:10.

[21] B. K. Aichernig, H. Brandl, E. Jöbstl, and W. Krenn, "Model-based Mutation Testing of Hybrid Systems," in *Formal Methods for Components and Objects - 8th International Symposium, FMCO 2009, Eindhoven, The Netherlands, November 4-6, 2009. Revised Selected Papers*, ser. Lecture Notes in Computer Science, F. S. de Boer, M. M. Bonsangue, S. Hallerstede, and M. Leuschel, Eds., vol. 6286. Springer-Verlag, 2010, pp. 228–249.

[22] K. G. Larsen, F. Lorber, B. Nielsen, and U. M. Nyman, "Mutation-based test-case generation with ecdar," in *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, March 2017, pp. 319–328.

[23] C. Furia, D. Mandrioli, A. Morzenti, and M. Rossi, *Modeling Time In Computing*. Springer-Verlag, 2012.

[24] R. Alur and D. L. Dill, "A theory of timed automata," *Theor. Comput. Sci.*, vol. 126, no. 2, pp. 183–235, 1994. [Online]. Available: https://doi.org/10.1016/0304-3975(94)90010-8

[25] A. David, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski, "Timed i/o automata: A complete specification theory for real-time systems," in *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control*, ser. HSCC '10. New York, NY, USA: ACM, 2010, pp. 91–100. [Online]. Available: http://doi.acm.org/10.1145/1755952.1755967

[26] C. Norström and A. Wall, "Timed automata as task models for event-driven systems," in *In proceedings of RTCSA99*. IEEE Computer Society, December 1999. [Online]. Available: http://www.es.mdh.se/publications/69-

[27] S. C. P. F. Fabbri, J. C. Maldonado, T. Sugeta, and P. C. Masiero, "Mutation testing applied to validate specifications based on statecharts," in *Proceedings of the 10th International Symposium on Software Reliability Engineering*, ser. ISSRE '99. Washington, DC, USA: IEEE Computer Society, 1999, pp. 210–. [Online]. Available: http://dl.acm.org/citation.cfm?id=851020.856195

[28] J. Tretmans, "Test generation with inputs, outputs and repetitive quiescence," *Software-concepts and tools*, vol. 17, no. 3, pp. 103–120, 1996.

[29] R. Nilsson and J. Offutt, "Automated testing of timeliness: A case study," in *Proceedings of the Second International Workshop on Automation of Software Test*, ser. AST '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 11–. [Online]. Available: http://dx.doi.org/10.1109/AST.2007.5

[30] R. de Simone, J.-P. Talpin, and D. Potop-Butucaru, *Embedded Systems Handbook*. CRC Press, 2005, audiocd The Synchronous Hypothesis and Synchronous Languages.

[31] N. T. H. Quyen, K. T. Tung, L. T. M. Hanh, and N. Thanh Binh, "Improving mutant generation for simulink models using genetic algorithm," in *Conference on Electronics, Information, and Communications (EICC)*, 2016.

[32] L. T. M. Hanh, K. T. Tung, and N. T. Binh, "Mutation-based test data generation for simulink models using genetic algorithm and simulated annealing," *Journal of Computer and Information Technology*, vol. 3, no. 4, pp. 763–771, 2014.

[33] Y. Yong Feng, Z. Yi Bin, and W. Yan Rong, "Research and improvements on mutation operators for simulink models," *Applied Mechanics and Materials*, vol. 687-691, pp. 1389–1393, 2014.

[34] M. Stephan, M. Alalfi, and J. Cordy, "Towards a taxonomy for simulink model mutations," in *Workshops of the International Conference on Software Testing, Verification and Validation (ICSTW)*, 2014.

[35] M. Stephan, "Model clone detector evaluation using mutation analysis," in *Conference on Software Maintenance and Evolution*, 2014.

[36] Y. Zhan and J. Clark, "Search-based mutation testing for simulink models," in *Conference on Genetic and Evolutionary Computation (GeCCO)*, 2005.

[37] A. Brillout, N. He, M. Mazzucchi, D. Kröning, M. Purandare, P. Rümmer, and G. Weissenbacher, "Mutation-based test case generation for simulink models," in *Formal Methods for Components and Objects*, 2010, pp. 208–227.

[38] N. He, P. Rümmer, and D. Kröning, "Test-case generation for embedded simulink via formal concept analysis," in *Design Automation Conference*, 2011, pp. 224–229.

[39] R. F. Araujo, J. C. Maldonado, M. E. Delamaro, A. M. R. Vincenzi, and F. Delebecque, "Devising mutant operators for dynamic systems models by applying the hazop," in *International Conference on Software Engineering Advances*, 2011.

[40] H. Runge, "A mutation analysis framework for simulink models," Master's thesis, Mälardalens Högskola, 2018.

[41] H. V. Do, C. Robach, and M. Delaunay, "Mutation analysis for reactive system environment properties," in *Workshop on Mutation Analysis*, 2006.

[42] L. Du Bousquet and M. Delaunay, "Using mutation analysis to evaluate test generation strategies in a synchronous context," in *International Conference on Software Engineering Advances (ICSEA)*, 2007.

[43] V. Papailiopoulou, "Automatic testing of lustre/scade programs. (test automatique de programmes lustre/scade)," Ph.D. dissertation, Joseph Fourier University, Grenoble, France, 2010. [Online]. Available: https://tel.archives-ouvertes.fr/tel-00454409

[44] R. H. Untch, A. J. Offutt, and M. J. Harrold, "Mutation analysis using mutant schemata," in *ISSTA*, 1993, pp. 139–148.

[45] X. Devroey, G. Perrouin, M. Papadakis, P.-Y. Schobbens, and P. Heymans, "Featured Model-based Mutation Analysis," in *International Conference on Software Engineering, ICSE*. Austin, TX, USA: ACM, 2016.

[46] P. Ammann and J. Offutt, *Introduction to software testing*. Cambridge University Press, 2008.

[47] M. Papadakis, Y. Jia, M. Harman, and Y. Le Traon, "Trivial compiler equivalence: A large scale empirical study of a simple fast and effective equivalent mutant detection technique," in *International Conference on Software Engineering, ICSE*. IEEE, 2015, pp. 936–946.

[48] X. Devroey, G. Perrouin, M. Papadakis, A. Legay, P.-Y. Schobbens, and P. Heymans, "Model-based mutant equivalence detection using automata language equivalence and simulations," *Journal of Systems and Software*, pp. –, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121218300475

[49] K. Cerāns, "Decidability of bisimulation equivalences for parallel timer processes," in *Proceedings of the 4th International Workshop on Computer Aided Verification (CAV'92)*, ser. Lecture Notes in Computer Science, G. von Bochmann and D. K. Probst, Eds., vol. 663. Springer-Verlag, 1993, pp. 302–315.

[50] R. Alur, C. Courcoubetis, and T. A. Henzinger, "The observational power of clocks," in *Proceedings of the 5th International Conference on Concurrency Theory (CONCUR'94)*, ser. Lecture Notes in Computer Science, B. Jonsson and J. Parrow, Eds., vol. 836. Springer-Verlag, Aug. 1994, pp. 162–177.

[51] F. Siavashi, J. Iqbal, D. Truscan, and J. Vain, *Testing Web Services with Model-Based Mutation*. Springer, 2017, vol. 743, p. 45–67.

[52] G. Behrmann, A. David, and K. G. Larsen, "A tutorial on UPPAAL," in *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004*, ser. LNCS, M. Bernardo and F. Corradini, Eds., no. 3185. Springer–Verlag, September 2004, pp. 200–236.

[53] M. Gligoric, S. Khurshid, S. Misailovic, and A. Shi, "Mutation testing meets approximate computing," in *International Conference on Software Engineering, NIER*, 2017, pp. 3–6.

[54] F. Hariri, A. Shi, O. Legunsen, M. Gligoric, S. Khurshid, and S. Misailovic, "Approximate Transformations as Mutation Operators," in *11th IEEE Conference on Software Testing, Validation, and Verification*, 2018.

[55] J. J. Ortiz, M. Amrani, and P. Schobbens, "Multi-timed bisimulation for distributed timed automata," in *NASA Formal Methods - 9th International Symposium, NFM 2017, Moffett Field, CA, USA, May 16-18, 2017, Proceedings*, ser. Lecture Notes in Computer Science, C. Barrett, M. Davies, and T. Kahsai, Eds., vol. 10227, 2017, pp. 52–67. [Online]. Available: https://doi.org/10.1007/978-3-319-57288-8

[56] D. Budgen and P. Brereton, "Performing systematic literature reviews in software engineering," in *Proceedings of the 28th international conference on Software engineering*. ACM, 2006, pp. 1051–1052.