Verifying Transient Behavior Specifications in Chaos Engineering Using Metric Temporal Logic and Property Specification Patterns

Sebastian Frank
University of Hamburg
Hamburg, Germany
sebastian.frank@uni-hamburg.de

Alireza Hakamian
University of Stuttgart
Stuttgart, Germany
mir-alireza.hakamian@iste.uni-stuttgart.de

Denis Zahariev
University of Stuttgart
Stuttgart, Germany

André van Hoorn
University of Hamburg
Hamburg, Germany
andre.van.hoorn@uni-hamburg.de

ABSTRACT

Chaos Engineering is an approach for assessing the resilience of software systems, i.e., their ability to withstand unexpected events, adapt accordingly, and return to a steady state. The traditional Chaos Engineering approach only verifies whether the system is in a steady state and considers no statements about state changes over time and timing. Thus, Chaos Engineering conceptually does not consider transient behavior hypotheses, i.e., specifications regarding the system behavior during the transition between steady states after a failure has been injected. We aim to extend the Chaos Engineering approach and tooling to support the specification of transient behavior hypotheses and their verification.

We interviewed three Chaos Engineering practitioners to elicit requirements for extending the Chaos Engineering process. Our concept uses Metric Temporal Logic and Property Specification Patterns to specify transient behavior hypotheses. We then developed a prototype that can be used stand-alone or to complement the established Chaos Engineering framework Chaos Toolkit. We successfully conducted a correctness evaluation comprising 160 test cases from the Timescales benchmark and demonstrate the prototype's applicability in Chaos Experiment settings by executing three chaos experiments.

CCS CONCEPTS

• **Software and its engineering → Software fault tolerance**; *Software testing and debugging*; *Formal software verification.*

KEYWORDS

chaos experiments, resilience, transient behavior, metric temporal logic, property specification patterns

ICPE '23 Companion, April 15–19, 2023, Coimbra, Portugal

ACM ISBN 979-8-4007-0072-9/23/04…$15.00
https://doi.org/10.1145/3578245.3584314

ACM Reference Format:
Sebastian Frank, Alireza Hakamian, Denis Zahariev, and André van Hoorn. 2023. Verifying Transient Behavior Specifications in Chaos Engineering Using Metric Temporal Logic and Property Specification Patterns. In *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering (ICPE '23 Companion), April 15–19, 2023, Coimbra, Portugal.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3578245.3584314

1 INTRODUCTION

Modern software systems should be resilient, i.e., behave properly despite disturbances and recover from service degradation [32]. A widespread approach for resilience engineers to assess their software systems' resilience is Chaos Engineering (CE) [5], i.e., continuous experimentation by continuously executing so-called chaos experiments. A chaos experiment consists of two crucial elements: a steady-state hypothesis and a fault injection. The steady-state hypothesis is usually evaluated twice, i.e., at an experiment's start and end. CE tools like Chaos Toolkit (CTK) [24] also allow for testing the *steady-state* hypothesis repeatedly during the experiment. However, even then, they only consider the system state at the time of the hypothesis evaluation. As shown in previous work [14], software architects are also interested in analyzing *transient behavior*, i.e., the behavior during the transition between steady states after a failure has been injected. For example, in a quality scenario [6, 14], the expected system response "autoscaling helps" could be refined to "when the response time exceeds a certain limit, the autoscaler should (repeatedly) act until the response time is below the threshold". CE conceptually does not allow the specification of such an expected response measure as transient behavior hypotheses.

We aim to study how the CE process and tooling can be extended to allow verifying transient behavior hypotheses. To reach our goal, we extend the CE process with a transient behavior hypothesis using Metric Temporal Logic (MTL) [18] as the underlying formalism, as it allows specifying transient behavior consisting of states, order, and timing behavior. Consequently, we apply runtime verification [19] of the MTL formulas based on monitoring data.

Our extended CE approach also aims to be helpful to CE practitioners. Hence, we conducted interviews with three CE practitioners. Considering their feedback, we designed the approach and prototype shown in Figure 1 that supports CE practitioners in (i) specifying, (ii) verifying, and (iii) investigating transient behavior. We also support Property Specification Patterns (PSP) [2]

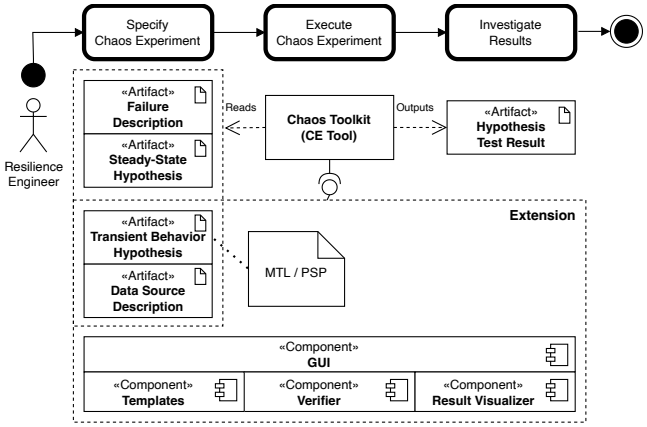

Figure 1: Extended Chaos Engineering Process

as an easier-to-use formalism than MTL. Regarding the result investigation, our concept incorporates visualizing monitoring data alongside color highlighting for verification results. The prototype is loosely coupled to the state-of-the-art CE framework Chaos Toolkit [24] and can also be used stand-alone.

We conduct a correctness evaluation with data from the Timescales benchmark [31] for MTL monitoring tools yielding 160 tests, covering four PSP types, positive and negative outcomes, and Future-MTL and Past-MTL formulas. We design and execute three chaos experiments on a mock of an industrial system and verify transient behavior hypotheses using our prototype. The prototype verified all benchmark formulas and transient behaviors correctly.

To summarize, our contributions presented in this work are:

- The requirements and expectations from interviews with CE practitioners regarding an extended CE approach.
- A concept and browser-based prototype extending CE for specifying, verifying, and investigating transient behavior.
- Supplementary material [33, 34] including the three chaos experiments. Note that we cannot provide the mock of the industrial system for confidentiality reasons.

The remainder of this paper is structured as follows: Section 2 introduces MTL, PSP, and CTK. Section 3 summarizes related works. The conducted interviews are presented in Section 4. Section 5 presents our concept, developed based on the practitioners' feedback. Subsequently, Section 6 describes the developed prototype, and Section 7 outlines the conducted evaluation. Finally, Section 8 summarizes the work and mentions potential future work.

2 BACKGROUND

In the following, we introduce MTL and PSP, which our approach uses as formalisms to describe transient behavior. Furthermore, we introduce CTK, the CE tool extended by our prototype.

2.1 Metric Temporal Logic

Metric Temporal Logic (MTL) [18] extends Linear Temporal Logic (LTL) [23] temporal operators by time intervals. Additionally, MTL supports past and future operators. MTL is often split semantically into Past-MTL and Future-MTL depending on the utilized temporal operators. Given a finite set P of atomic propositions, Past-MTL can be defined by the following grammar [28]:

$$\varphi = \top \mid p \mid \neg\, \varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1\, S_I\, \varphi_2$$

where $p \in P$ and $I \subseteq [0, \infty)$. The subscript I represents a time interval, i.e., the temporal operator's timing constraint. Further operators can be derived, e.g., $\wedge$ (and), $\blacklozenge_I$ (once), and $\blacksquare_I$ (always). For example, the MTL formula "$\blacklozenge_{[0,5]} Q$" means that Q once was satisfied within the last 5 time units (TU). Future-MTL comprises equivalent operators evaluated in the future direction instead of the past, i.e., $\Box_I$ (always), $\Diamond_I$ (eventually), and U_I (until) [3].

MTL formulas can contain predicates, also called propositional functions. They are expressions that contain variables and become statements when the variables are substituted with constants [11].

2.2 Property Specification Patterns

Property Specification Patterns (PSP) [13] are recurring patterns that can be mapped to (structured) natural language [2] or temporal logics, e.g., LTL or MTL. For example, the *response* pattern "Globally, event P must always be followed by event Q within 5 s" can be translated to Future-MTL as "$\Box(P \rightarrow \Diamond_{[0,5]} Q)$". Each pattern also has one of the five scopes "Globally", "Before Q", "After Q", "Between Q and R", "After Q until R".

Czepa and Zdun [12] found evidence that PSP are more comprehensible than LTL. Furthermore, Autili et al. [2] introduced the tool PSP Wizard, allowing its users to easily create PSP specifications in natural language and translate them to various temporal logics.

2.3 Chaos Toolkit

Chaos Toolkit (CTK) [24] is an open-source tool for executing automated chaos experiments [5]. It has a variety of extensions that allow the injections of faults on the platform, network, and application layer. CTK supports chaos experiments specified by the YAML[1] data serialization language. CTK's chaos experiments consist of a steady-state hypothesis and a fault injection called the method. A steady-state hypothesis usually contains probes that retrieve information from the system under test, while the method usually contains actions that perform operations on the system under test.

3 RELATED WORK

We investigated various CE tools, i.e., Chaos Mesh [25], Gremlin [16], ChaosBlade [10], CTK [24], Litmus [20], CloudStrike [27], and Chaos Monkey [21]. The tools distinctly differ in the targeted quality attributes, supported platforms, and considered system layer. Nevertheless, none of these tools supports transient behavior hypotheses. In particular, they do not use MTL or PSP. Some tools, e.g., CTK [24], allow repeatedly testing the steady-state hypothesis during the experiment. While this enables simple transient behavior analysis, it lacks the expressiveness of MTL regarding timings.

There are applications of temporal logic and runtime verification in the more general field of resilience engineering. The tool Lotus@Runtime [4] constructs a probabilistic system model from monitoring data. However, it only allows specifying reachability statements and not complex temporal behavior. Cámara and De Lemos [8] propose an approach for verification of self-adaptive

[1] https://yaml.org/

systems and resilience properties based on probabilistic model-checking. Collected data from stimulating the system's environment is transformed into Discrete-Time Markov Chains (DTMCs). DTMCs are a probabilistic approach used to generalize the results of multiple executions, while CE is about single runs. Thus, they use model checking while we use runtime verification. The authors also specify resilience properties using Probabilistic Computation Tree Logic (PCTL) expressions based on the *response* pattern from PSP. However, in general, PSP are not part of their approach, and they use the *response* pattern to describe both the stimulus and the expected response, which is still a steady-state hypothesis.

The work by Bursztein and Goubault-Larrecq [7] uses a variant of Timed Alternating-Time Temporal Logic (TATL) to specify resilience properties but focuses specifically on network resilience. Runtime verification is also applied by Torjusen et al. [26] for specifying self-adaptive security and privacy properties. However, their approach is specific to the IoT and health domain. Similarly, the work by Gouglidis et al. [15] specifies resilience properties using Computation Tree Logic (CTL) but tests against resilience policies for access control. An approach by Catano [9] aims to generate code for safety and mission-critical systems to satisfy resilience properties expressed in LTL. However, it does not consider timing behavior as in MTL. Furthermore, full code generation is usually infeasible for non-safety-critical systems as targeted by CE.

4 REQUIREMENTS ELICITATION

The interviews aimed to explore whether CE practitioners have examined transient behaviors and have experienced difficulties while specifying transient behaviors. Furthermore, the interviews aimed to gather requirements regarding the extended CE approach and feedback on our initial concepts and ideas for its realization.

4.1 Setting and Execution

We invited seven industry and academic employees with CE experience. Our invitation comprised a short interview description and a consent form. Four people answered with interest in participating. However, one was not able to schedule an interview. The remaining three participants comprised (i) a senior software architect from industry, (ii) a master's student and research assistant who has done CE research, and (iii) a master's student involved in CE research and applying CE in the industry.

The interviews consisted of 20 questions in three parts. In the first part, we gave a short introduction and asked the interviewees about their expertise and preceding attempts to analyze transient behavior. In the second part, we asked about the interviewees' opinions on using concepts such as MTL, PSP, and runtime verification. Further, we gathered feedback on their potential use of our approach and specific suggestions. A video demonstration of an early version of our prototype accompanied the questions. In the third part, we elicited the interviewees' requirements regarding the extended CE approach using a question catalog. The catalog was split into conceptual, technical, and quality requirements.

We defined the following hypotheses that reflect the initial design decisions of our concept and development plans:

- H_1: The interviewees have experienced situations where they could not define behaviors they wanted to verify.
- H_2: The interviewees have tried to analyze behaviors unrelated to the steady state.
- H_3: The interviewees agree that the proposed CE extension provides deeper insights into software systems' resilience and view it as meaningful.
- H_4: The interviewees want to verify transient behavior hypotheses independent of CE, i.e., in a stand-alone analysis.
- H_5: The concept of specifying transient behavior hypotheses using MTL and PSP is sufficient and complete.
- H_6: The concept of verifying transient behavior hypotheses using runtime verification is sufficient and complete.
- H_7: A visualization containing the verification results and the monitoring data is a helpful output of the approach.
- H_8: Usability, extensibility, and performance are important quality attributes for the proposed approach and prototype.

The interviews were conducted as video calls and in a semi-structured manner. Each interview took 60 min to 90 min. Additionally, all interviews were recorded with the participants' consent.

4.2 Results

All three interviewees stated they have experience with CE. However, two said that they would lack practical experience. All interviewees have experienced situations where they wanted to verify non-steady-state behaviors, and two were able to verify them. However, one had to do much manual work, and the third interviewee could achieve this only partially. Thus, the participants confirm H_1 and, to some extent, H_2, supporting the relevance of our work.

All participants agreed that the proposed approach would provide deeper insights into the resilience of software systems, confirming H_3. Regarding how to adopt the approach, one interviewee could not answer, as he was not actively practicing CE back then. Another interviewee responded that some interfaces or adapters would be required to integrate the extended CE approach into his workflow. The last interviewee answered that integration into his workflow would be easy since he uses similar technologies.

Two interviewees would regularly use the proposed approach if their chaos experiments had some defined criteria regarding transient behaviors. The third interviewee would use the approach when he wants to know more about the system than whether or not a particular service is still running. Additionally, two interviewees said they could imagine using the approach and the prototype for scenarios and use cases outside the scope of CE, which confirms H_4 and emphasizes the importance of a stand-alone feature.

All (somewhat) agreed that the concept of specifying transient behaviors is complete and sufficient. However, two were concerned about the difficulty in obtaining meaningful values for the specifications. Additionally, two participants said that some assistance in creating the transient behavior specifications would be helpful. Confirming H_5, all interviewees opted for both presented input options, i.e., MTL and PSP. The participants appreciated MTL's expressiveness and PSP's readability and practicability, especially when working with specifications they did not create. They suggested additions to the behavior specification, e.g., more logical predicates, such as increasing and decreasing trends. Regarding supported monitoring tools, the interviewees suggested CSV files, time series databases in general, and Prometheus [1] in particular.

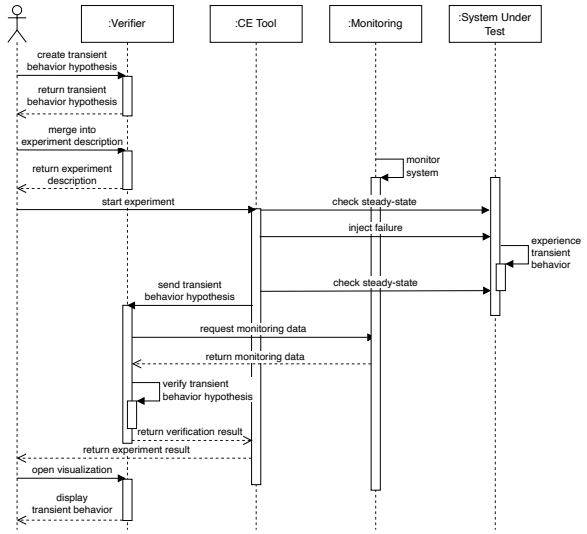

Figure 2: Interaction between our extended CE approach ("Verifier") and its environment

Regarding H_6, the interviewees mostly agreed that the concept for verifying hypotheses is complete and sufficient. However, they stated that more information in the verification's output would be helpful, e.g., which parts of a hypothesis failed. For this reason, two interviewees liked the idea of visualizing monitoring data and verification results, which was not very important to the third participant. This still rather supports H_7. Two participants denoted the discrete nature of the event traces a limitation. The third interviewee said that in practice, it is a strength of the approach since he is unaware of monitoring tools capturing measurements in non-discrete ways.

An interviewee called usability the prototype's essential quality, and other features could be neglected in its favor. Two interviewees stated that a user interface would increase the prototype's usability. Furthermore, all interviewees ranked extensibility as important. However, two ranked response times as of minor importance because event traces should be short as transient behaviors typically hold over a short duration, i.e., minutes. This partially confirms H_8.

5 CONCEPT

The proposed CE extension comprises (i) a format for specifying transient behavior, (ii) a method for verifying the specified transient behavior, and (iii) means to present the verification results to the users. The approach and its three parts are detailed in the following.

5.1 Overview

We define the extended CE process as follows: (1) build a steady-state hypothesis, (2) build a transient behavior hypothesis, (3) vary real-world events, (4) run experiments in production, (5) automate experiments, and (6) minimize blast radius. We added step 2 to the original process by Basiri et al. [5] while keeping the other steps.

Based on the extended CE process and the elicited requirements (see Section 4), we derived the potential interaction of a resilience engineer using our approach, as displayed in Figure 2. First, the resilience engineer has to specify a transient behavior hypothesis. Our prototype can assist in formulating and integrating this hypothesis into the chaos experiment description. Then, the initial steady-state check and the anomaly injection are executed as in the traditional CE approach. After the experiment and the second steady-state check, the transient behavior check is initiated. The CE tool sends the transient behavior hypothesis to the prototype. Then, our prototype collects the required monitoring data regarding the transient phase from the system's monitoring. Next, the verification is performed, and the CE tool receives the verification result, i.e., whether the transient behavior hypothesis holds. Finally, the resilience engineer can inspect the monitoring data and results in visualizations provided by the prototype.

Our process relies on two assumptions: First, monitoring in the system under test is in place, and monitoring data is recorded with sufficient accuracy, e.g., in a time-series database. Second, the verification is executed offline, i.e., at the experiment's end. This is because transient behavior hypotheses could get satisfied later in the experiment, e.g., when the response Q of the *response* pattern could still occur. However, sometimes hypotheses cannot be satisfied any longer, e.g., when Q's time limit is reached. Then, the experiment could be aborted. Thus, we consider detecting such irreversible states a valuable direction for future work.

5.2 Transient Behavior Specification

Traditionally, a steady-state hypothesis is a statement regarding a single system measurement at a certain time, verified against a certain tolerance [5]. Even if the steady-state hypothesis is carried out repeatedly, it is still a statement regarding only a single point in time. In contrast, a transient behavior hypothesis considers a series of points in time or even the entire experiment. Therefore, we decided to use MTL [18] as the underlying formalism for describing transient behavior since it can express states' occurrence, order, and timing behavior. For example, $\square(responsetimehigh \rightarrow \Diamond_{[0,30]}(scaledup \vee \neg responsetimehigh))$ describes that whenever the response time is high, within 30 TU, the autoscaler should spawn a new instance, or the response time is not high any longer.

As explicitly stated by the interviewees, specifications in MTL are hard to provide without any experience in temporal logic. Thus, our extended CE approach supports PSP as an abstraction layer for the complexity of MTL. For example, the previous example in MTL could be expressed by the *response* pattern as "Globally, if (*responsetimehigh*) [has occurred] then in response (*scaledup or not responsetimehigh*) [eventually holds] within 30 TU".

In addition, since the logical variables contained in an MTL formula or a PSP definition do not include information regarding how they should be interpreted, a transient behavior hypothesis must also contain information concerning the logical variables, events, and predicates featured in the behavior description. Our approach defines eleven basic predicate types described in Table 1. Note that custom predicates can be handled by pre-evaluating the measurements and using the boolean predicate type in the approach.

Listing 1 depicts how the specification concept could be implemented in the JavaScript Object Notation (JSON) format. We decided on a data language because CTK uses a data language as well, which is suitable for representing configuration information. Line 2

Table 1: Supported default predicates $P(x_t)$ with x_t being the measurement for time t and c a constant

type	operator
boolean	$x_t = c$
equal	$x_t = c$
notEqual	$x_t \neq c$
bigger	$x_t > c$
biggerEqual	$x_t \geq c$
smaller	$x_t < c$
smallerEqual	$x_t \leq c$
trendUpward	$x_t \geq x_{t-1}$
trendUpwardStrict	$x_t > x_{t-1}$
trendDownward	$x_t \leq x_{t-1}$
trendDownwardStrict	$x_t < x_{t-1}$

shows the transient behavior description for the previously introduced example. Information about the used formalism (lines 3 & 4) is also provided to the approach. In this case, the description uses keywords of Past-MTL, but they are interpreted as their equivalent Future-MTL keywords (see Section 6.1).

After that, further building blocks of the hypothesis are defined, i.e., predicates and data sources. In the example, the hypothesis contains two predicates, i.e., *scaledup* evaluates to true when a data point value equals one (lines 7-9), and *responsetimehigh* evaluates to true when the provided data point value is higher than one. The data sources are defined as CSV file columns (line 17). Each predicate is associated with one data set, e.g., the *rt* data set is associated with the *response_time* column.

```
1  "specification": "always((responsetimehigh(rt)) -> (
       once[0,30] (scaledup(scaling1) or scaledup(
       scaling2) or (not responsetimehigh(rt))))",
2  "specification_type": "mtl",
3  "future-mtl": "true",
4  "predicates_info": [
5      {
6          "predicate_name": "scaledup",
7          "predicate_logic": "equal",
8          "predicate_comparison_value": "1"
9      },
10     {
11         "predicate_name": "responsetimehigh",
12         "predicate_logic": "bigger",
13         "predicate_comparison_value": "1.0"
14     }
15 ],
16 "measurement_source": "csv",
17 "measurement_points": [
18     {
19         "measurement_name": "rt",
20         "measurement_column": "response_time"
21     },
22     {
23         "measurement_name": "scaling1",
24         "measurement_column": "scaling_ex1"
25     },
26     {
27         "measurement_name": "scaling2",
28         "measurement_column": "scaling_ex2"
29     }
30 ]
31 }
```

Listing 1: Example of a transient behavior hypothesis for the expected response "autoscaling helps" in the JSON format.

5.3 Verification

Since CE requires verifying individual experiment runs, it is reasonable to use runtime verification [19] as a more lightweight approach to verify transient behavior hypotheses compared to heavyweight model checking. That means, most notably, no formal model of the system must be created, but monitoring needs to be in place. However, the use of PSP allows transformations to various formal languages suitable for model checking, which we consider an extension of the approach in future work.

The concept assumes that a monitoring approach monitors relevant events in a discrete-time manner. Moreover, we assume that the relevant monitoring data can be retrieved from a time-series database that persists all the relevant monitoring data. Therefore, the verification approach requires a fitting query to obtain the desired data. Furthermore, since monitoring tools provide data in different formats and ways, our concept foresees the CSV format as a generic format to represent monitoring data. This also contributes to the approach's requirement to be usable stand-alone, i.e., without integration into specific CE or monitoring tools.

5.4 Assistance

Based on the interviews, two activities in working with transient behavior hypotheses in CE require assistance: (1) providing a transient behavior specification and (2) investigating the results.

Besides using PSP, our concept contains the (optional) use of a wizard to ease the provision of transient behavior hypotheses. A wizard helps the resilience engineer to see (1) which parts of the specification are still missing and (2) which elements are supported.

The verification result is a concrete logical value in the traditional CE approach. As elicited, additional information should be available, such as when the result changes from one value to another, e.g., from true to false, and the values that caused this change. Including these additional outputs provides the reasoning behind the final verification result and makes the outputs of the extended approach more informative. We propose to use a line chart for the visualization that displays the event traces used during the verification presented as lines and verification results displayed as background colors of the respective section in the plot.

6 PROTOTYPE

We have prototypically implemented our concept for specifying and verifying transient behavior in CE as a web-based tool that can easily be deployed in a Docker[2] container. The prototype can either run as a stand-alone application, i.e., to specify, verify, and investigate transient behavior by manually providing the necessary data or as a loosely coupled extension for CTK [24]. In the following, we present the characteristics and limitations of the prototype regarding the specification, verification, user interface, CTK integration, monitoring integration, and architecture.

[2]https://www.docker.com

6.1 Specification & Verification Capabilities

The prototype can import transient behavior hypotheses in the JSON format, e.g., such as in Listing 1. For verifying Past-MTL formulas, the prototype uses a Python library [29]. Support for Future-MTL is achieved by using the Past-MTL operators but reverting the monitoring trace, which is possible due to the traces' limited size. However, the provided plots for result investigation only provide limited functionality when using this feature. All predicates introduced in Section 5.2 are supported. However, only PSP for which the work by Ulus [31] provides a definition in past-MTL are currently implemented, i.e., 15 variants of the *absence*, *universality*, *response*, and *recurrence* patterns. Note that PSP equivalents mapped to Future-MTL can be imported.

6.2 Graphical User Interface

The prototype provides a Graphical User Interface (GUI) that can be accessed using a web browser and consists of seven web pages. Two pages are for navigating to the remaining five functional pages. On the MTL and the PSP page, users can select the building blocks for the hypothesis and the measurement source using drop-down menus with pre-configured choices. On another page, users can place an already defined hypothesis into a CTK experiment description. This page comprises a text box where the JSON-formatted hypothesis must be pasted and a file input field for the experiment description. Similarly, another page allows users to verify the transient behavior hypothesis stand-alone. It comprises a simple text box for the hypothesis and a file input field for the CSV-formatted monitoring data. Finally, a page displays the plot for investigation of the verification results.

6.3 Chaos Toolkit Integration

We focused on extending CTK as it is an open-source, state-of-the-art tool that allows injecting failures on the platform, network, and application layers. Furthermore, it follows the CE process [5] and uses the HTTP protocol for communication, which allows extending it easily, as we have done it already in previous work [14].

Chaos experiments in CTK are YAML files compatible with the JSON format of our transient behavior hypotheses. Therefore, using the prototype's GUI, the resilience engineer can easily add the hypothesis to CTK's experiment description. Technically, this is realized as a probe in the experiment's steady-state hypothesis.

When the steady state hypothesis is verified at the end of the experiment, the probe sends an HTTP request containing the transient behavior hypothesis to the prototype. The prototype will then obtain the monitoring data and verify the results. Its reply indicates whether the hypothesis was fulfilled or not. CTK can then report the result for all hypotheses.

6.4 Monitoring Integration

The prototype can obtain monitoring data from the time-series databases InfluxDB [17] and Prometheus [1] provided that the location of the database is known. While the monitoring needs to be configured separately, queries can be defined for the prototype to access the required data. For example, the query *SELECT \"counts\" FROM \"AliveInstances\".\"autogen\".\"InstanceCounts\"* can be used to get the number of a service's instances when using InfluxDB.

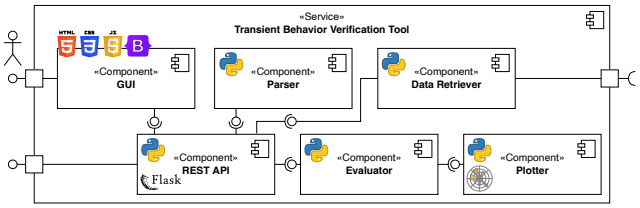

Figure 3: Architecture of the prototype

The prototype supports CSV files for uploaded experiment data or interoperability with other tools. It assumes that each row represents the system state at a certain time. Time units are interpreted implicitly, meaning the resilience engineer must consistently use the same time units in the specification and monitoring.

6.5 Architecture

The prototype is implemented in Python and can be deployed in a Docker container. An overview of the architecture comprising six components is presented in Figure 3. The code is hosted in GitHub [34] including several screenshots of the user interface. The following describe the architecture in more detail.

To enable communication via the HTTP protocol, a REST API is implemented using the web framework Flask[3]. The API provides several endpoints. However, the most important one is the "/monitor" endpoint. This endpoint listens for HTTP requests containing transient behavior hypotheses as JSON objects, and when such a request is received, the verification of the received hypotheses is initiated. The REST API is also used to serve the web pages comprising the GUI, which is created using HTML, CSS, JavaScript, the Bootstrap framework[4], and Flask's template engine Jinja[5].

The parser transforms PSP to MTL formulas and can interpret predicates. The data retrievers take care of collecting the required monitoring data. There are data retrievers for InfluxDB, Prometheus, and CSV files. The obtained data is then provided to the evaluator component, which is responsible for verifying the prepared MTL formula against monitoring data using a Python library [30]. Finally, the plotter creates plots showing the monitoring data, the verification results, and the interval information using matplotlib[6].

7 EVALUATION

We conducted two types of evaluations, one regarding the correctness of the transient behavior hypotheses verification and another regarding the approach's applicability in CE settings.

7.1 Verification Correctness

In the first evaluation, we investigate whether the prototype can correctly verify Future- and Past-MTL formulas, i.e., the prototype returns the correct verification result for pairs of MTL formulas and monitoring traces. Additional data and experiment results can be found in the supplementary material [33].

[3] https://flask.palletsprojects.com
[4] https://getbootstrap.com
[5] https://jinja.palletsprojects.com
[6] https://matplotlib.org

7.1.1 Experiment Description. We use a benchmarking tool for MTL monitoring tools called Timescales [31] to generate MTL formulas, monitoring traces, and the expected verification result. Even though Timescales primarily focuses on the performance evaluation of MTL monitoring tools, it can be used to evaluate the correctness of the verification. Timescales can provide MTL formulas for four PSP: *absence*, *universality*, *recurrence*, and *response* pattern. Since different scopes are supported, this results in 10 different MTL formulas available in Future- and Past-MTL variants. For each formula, one short (10 000 TU) and three large (1 000 000 TU) traces are available as CSV files. Furthermore, each of these traces can be generated to satisfy and not satisfy its corresponding MTL formula.

We generated all available test data combinations to evaluate our approach, resulting in 160 tests. The evaluation is executed using Postman[7], sending test data requests to the prototype. All generated tests exist in the supplementary material.

7.1.2 Results. All the 160 tests delivered the verification result as expected by the Timescales framework. Thus, we conclude that the prototype correctly verifies typical Past- and Future-MTL formulas.

7.2 Applicability in Chaos Engineering Settings

In this part of the evaluation, we investigate whether the prototype can verify transient behavior hypotheses in CE settings, i.e., when using CTK. Table 2 summarizes the three conducted experiments.

7.2.1 Experiment Description. To execute chaos experiments, we use a mock of a company's payment calculation system [14], also used by the company to evaluate its system. The actual business logic is not crucial for evaluating our approach's applicability. In previous research [14], we elicited resilience scenarios for the same system. The mocked system uses the microservice architecture style [22] and consists of four services. The first service, called *API-Gateway*, distributes the incoming load to two other services, *Service1* and *Service2*. Additionally, *Service1* can forward calls to Service2. A *Eureka* service is used for service discovery. Furthermore, the test system includes a load generator capable of generating a specific number of requests for the system's endpoints. The information regarding the generated load is additionally saved into an InfluxDB database instance. To enable the execution of chaos experiments on the test system, we deployed the services on a local microk8s[8] installation. The deployment was configured so that two instances of *Service1* and *Service2* were hosted on different pods.

We created and executed three chaos experiments using CTK and our prototype. The injected faults are a service instance crash in the first experiment and a sudden workload increase in the remaining two experiments. These fault types are based on the resilience scenarios we elicited in previous work [14]. The fault is injected by stopping a *Service1* pod while the system is in a normal mode of operation. Regarding the steady-state hypothesis, we expect the deployment to be fully available, i.e., all instances are running.

To evaluate the prototype's different features, the experiments cover MTL and PSP as input types and InfluxDB and Prometheus as monitoring data sources. We define the following three different transient behavior hypotheses: In the first experiment, we expect the system to spawn a new instance within 30 s without exceeding a response time of 100 ms when the number of instances of *Service1* drops below two. In the second experiment, we use the *response* pattern to express that the number of instances of *Service1* should be increased (above two) within 30 s to 60 s whenever the response time exceeds 150 ms. Our expectation in the third experiment is that the CPU utilization of both *Service1* pods is always below 80 %.

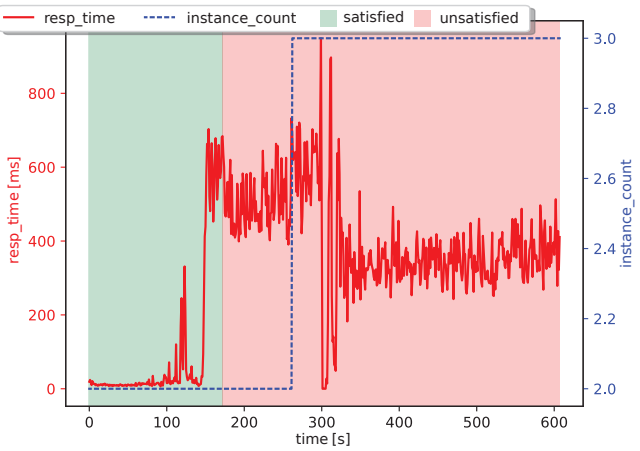

Figure 4: Result visualization for Chaos Experiment 2

7.2.2 Results. All experiments were conducted successfully, meaning no error was detected, and the prototype provided the correct results. We inspected the result in the visualization provided by the prototype to confirm the correctness of the verification.

In the first experiment, the transient behavior hypothesis is immediately violated since the response times increase to 250 ms after the instance failure. Figure 4 visualizes the result of the second experiment in which the first response time violation occurs at around 118 s. As a switch from the green to the red background color indicates, a new instance is not spawned in less than 60 s. Also, for the third experiment, the CPU utilizations of both pods violate the threshold of 80 %. In all three experiments, the prototype correctly detected the violation of the hypothesis.

7.3 Threats to Validity

The interviews with only three participants have a validity threat of low significance. However, we did not aim for exhaustive requirements elicitation or evaluation but for early feedback on crucial design decisions. Proper evaluation with more users is still required.

Regarding the evaluation of the verification correctness, there is the threat of not testing incorrectly implemented features. In particular, we did not test PSP inputs but only equivalent MTL formulas. However, an influence on the results is unlikely since the prototype internally translates PSP to MTL using established mapping rules [2]. We did not test patterns unavailable in the Timescales benchmark. However, the available patterns use all MTL operators. Thus, it is unlikely that errors arise in other patterns.

Regarding the evaluation of verification in chaos experiments, the main threat is that we did not execute chaos experiments representative of real-world chaos experiments. However, the mock

[7] https://www.postman.com
[8] https://microk8s.io

Table 2: Chaos experiments executed using CTK

	Chaos Experiment 1	**Chaos Experiment 2**	**Chaos Experiment 3**
Fault	Instance of Service1 crashes	Sudden workload increase	Sudden workload increase
Steady-State Hypothesis	Deployment fully available	Deployment fully available	Deployment fully available
Transient-State Hypothesis	After instance crash, spawn instance in <30 s; keep response time <100 ms	When response time exceeds 150 ms, spawn new instance within 30 s to 60 s	CPU usage of Service1 pods is always <80 %
Specification Type	Past-MTL formula	PSP	Past-MTL formula
Monitoring	InfluxDB	InfluxDB	Prometheus

system and the injected faults are taken from an industrial context. The transient behavior hypotheses are synthetic because a business stakeholder has not explicitly stated them. However, the stated hypotheses refer to common strategies in microservice-based systems [22], i.e., instance restarting (Chaos Experiment 1), autoscaling (Chaos Experiment 2), and load balancing (Chaos Experiment 3). Nevertheless, a user study is still necessary to investigate whether the prototype works according to the interest of actual users and to verify the satisfaction of the elicited requirements.

8 CONCLUSION

This work introduced our tool and concept for specifying transient behavior hypotheses using MTL and PSP in CE. It complements the steady-state hypothesis and allows considering more complex transient behavior. We interviewed CE practitioners to elicit their requirements regarding an extended CE approach and considered their feedback in the design, e.g., we added a GUI for specifying transient behavior and investigating the verification results. The prototype can be used with CTK or stand-alone. Our evaluation shows that the prototype verifies the hypotheses correctly for Timescales benchmark data and in CE settings using CTK.

In future work, we aim to extend the prototype with more complex (multi-parameter) predicates and add support for all PSP. We consider implementing additional features, e.g., support for time units, to increase usability. We also aim to utilize the prototype's stand-alone capabilities to verify simulated chaos experiments and for verification in an interactive scenario optimization process.

ACKNOWLEDGMENTS

The authors thank the German Federal Ministry of Education and Research (dqualizer FKZ: 01IS22007B and Software Campus 2.0 — Microproject: DiSpel, FKZ: 01IS17051) for supporting this work.

REFERENCES

[1] Prometheus Authors. 2023. Prometheus. https://prometheus.io
[2] Marco Autili, Lars Grunske, Markus Lumpe, Patrizio Pelliccione, and Antony Tang. 2015. Aligning qualitative, real-time, and probabilistic property specification patterns using a structured english grammar. *IEEE Transactions on Software Engineering* 41, 7 (2015), 620–638.
[3] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of model checking*. MIT press.
[4] Davi Monteiro Barbosa, Romulo Gadelha De Moura Lima, Paulo Henrique Mendes Maia, and Evilasio Costa. 2017. Lotus@ runtime: A tool for runtime monitoring and verification of self-adaptive systems. In *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 24–30.
[5] Ali Basiri, Niosha Behnam, Ruud Rooij, Lorin Hochstein, Luke Kosewski, Justin Reynolds, and Casey Rosenthal. 2016. Chaos Engineering. *IEEE* 33 (2016), 1–1.
[6] Len Bass, Paul Clements, and Rick Kazman. 2021. *Software Architecture in Practice* (4 ed.). Addison-Wesley Longman Publishing Co., Inc., USA.
[7] Elie Bursztein and Jean Goubault-Larrecq. 2007. A Logical Framework for Evaluating Network Resilience Against Faults and Attacks. In *Advances in Computer Science – ASIAN 2007. Computer and Network Security*, Iliano Cervesato (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 212–227.
[8] Javier Cámara and Rogério De Lemos. 2012. Evaluation of resilience in self-adaptive systems using probabilistic model-checking. In *2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 53–62.
[9] Nestor Catano. 2022. Program Synthesis for Cyber-Resilience. *IEEE Transactions on Software Engineering* (2022), 1–1.
[10] Cloud Native Computing Foundation. 2023. https://chaosblade.io
[11] Irving Copi, Carl Cohen, and Victor Rodych. 2016. *Introduction to logic*. Routledge.
[12] Christoph Czepa and Uwe Zdun. 2018. On the understandability of temporal properties formalized in linear temporal logic, property specification patterns and event processing language. *IEEE Transactions on Software Engineering* 46, 1 (2018), 100–112.
[13] Matthew B Dwyer, George S Avrunin, and James C Corbett. 1999. Patterns in property specifications for finite-state verification. In *Proceedings of the 21st international Conference on Software Engineering*. ACM, 411–420.
[14] Sebastian Frank, M. Alireza Hakamian, Lion Wagner, Dominik Kesim, Christoph Zorn, Jóakim von Kistowski, and André van Hoorn. 2021. Interactive Elicitation of Resilience Scenarios Based on Hazard Analysis Techniques. In *15th European Conference on Software Architecture (ECSA), Revised Selected Papers (Lecture Notes in Computer Science, Vol. 13365)*. Springer, 229–253.
[15] Antonios Gouglidis, Vincent C. Hu, Jeremy S. Busby, and David Hutchison. 2017. Verification of resilience policies that assist attribute based access control. In *Proc. of the 2nd ACM Workshop on Attribute-Based Access Control*. ACM, 43–52.
[16] Gremlin Inc. 2023. https://www.gremlin.com
[17] InfluxData Inc. 2023. InfluxDB. https://influxdata.com
[18] Ron Koymans. 1990. Specifying real-time properties with metric temporal logic. *Real-time systems* 2, 4 (1990), 255–299.
[19] Martin Leucker and Christian Schallhart. 2009. A brief account of runtime verification. *The journal of logic and algebraic programming* 78, 5 (2009), 293–303.
[20] LitmusChaos Authors. 2020. https://litmuschaos.io
[21] Netflix Inc. 2023. https://netflix.github.io/chaosmonkey
[22] Sam Newman. 2021. *Building microservices*. " O'Reilly Media, Inc.".
[23] Amir Pnueli. 1977. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. IEEE, 46–57.
[24] Chaos Toolkit Team. 2023. Chaos Toolkit. https://chaostoolkit.org
[25] The Linux Foundation. 2023. https://chaos-mesh.org
[26] Arild B Torjusen, Habtamu Abie, Ebenezer Paintsil, Denis Trcek, and Åsmund Skomedal. 2014. Towards run-time verification of adaptive security for IoT in eHealth. In *Proceedings of the 2014 European Conference on Software Architecture Workshops*. ACM, 1–8.
[27] Kennedy A Torkura, Muhammad IH Sukmana, Feng Cheng, and Christoph Meinel. 2020. Cloudstrike: Chaos engineering for security and resiliency in cloud infrastructure. *IEEE Access* 8 (2020), 123044–123060.
[28] Dogan Ulus. 2019. Online Monitoring of Metric Temporal Logic using Sequential Networks. *CoRR* (2019). arXiv:1901.00175 http://arxiv.org/abs/1901.00175
[29] Dogan Ulus. 2019. Pure Python package to monitor formal specifications over temporal sequences. https://github.com/doganulus/python-monitors.
[30] Dogan Ulus. 2019. Python-Monitors. https://github.com/doganulus/python-monitors
[31] Dogan Ulus. 2019. Timescales: A benchmark generator for MTL monitoring tools. In *International Conference on Runtime Verification*. Springer, 402–412.
[32] Katinka Wolter, Alberto Avritzer, Marco Vieira, and Aad van Moorsel. 2012. *Resilience Assessment and Evaluation of Computing Systems*. Springer.
[33] Denis Zahariev. 2022. Supplementary Materials. https://doi.org/10.5281/zenodo.6845089
[34] Denis Zahariev. 2023. GitHub Project. https://github.com/Cambio-Project/transient-behavior-verifier