CCaaS: Online Conformance Checking as a Service

Ingo Weber[1], Andreas Rogge-Solti[2], Chao Li[1], and Jan Mendling[2]

[1] NICTA, Sydney, Australia firstname.lastname@nicta.com.au
[2] Wirtschaftsuniversität Wien, Vienna, Austria, firstname.lastname@wu.ac.at

Abstract. Conformance checking, a method of process mining, is commonly used to assess how well a set of historic log traces fits a given process model, or vice versa. Here we explore online conformance checking, i.e., to check conformance on logs while they are written. This can be useful for near-realtime detection of errors and deviations from the desired path. While the online aspect leads to some challenges, we also study efficient detection of timing anomalies and violations of numerical invariants. The approach is implemented in CCaaS, a RESTful service that detects unfitting events and other errors in split seconds.

1 Introduction

One main activity in process mining is conformance checking, i.e., determining if a process model fits a given event log, or vice versa. Conformance checking is often used to assess the quality of discovery algorithms [1], or how well a model discovered from historic logs fits new logs. Thus conformance checking is routinely applied offline, and only subjected to event logs from completed traces.

In contrast, in this work we propose to use conformance checking as an online activity, to detect fitness errors as soon as they can be observed. Consider the example of earlier work, where we have shown that process-oriented analysis and online error detection can strongly improve system reliability [11]. In that work, process mining is used as a method to perform behavioral log analysis (through discovery) and tracking (through conformance checking) of systems.

The applicability of online conformance checking is much wider than system log analysis. Whenever a process model needs to be followed strictly, e.g., to ensure compliance with laws and regulations, online conformance checking can be used to detect deviations in near-realtime.

Our contributions in this paper are[4]:

- *Online Conformance Checking as a Service (CCaaS)*: whenever the RESTful CCaaS service is invoked with a new log event, it decides if the event is fit or unfit with a BPMN model.

 NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

[4] Conformance checking for processes with multi-instantiation has been explored in [7]. Our frontend, POD-Viz, was published as part of another demo paper [9].

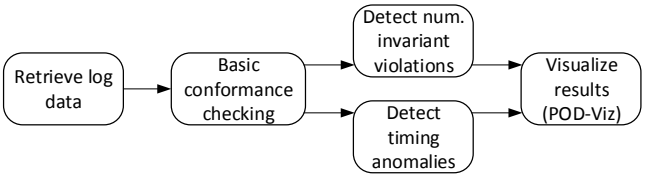

Fig. 1. CCaaS overview

- *Timing anomaly detection*, based on timing anomaly intervals that can be exported from a ProM plugin and imported into CCaaS, which in turn checks if the duration of any activity is anomalous.
- Applying *numerical invariants* to conformance checking: based on numerical information in log events, CCaaS learns how many loop iterations or multi-instantiated (MI) subprocess instances should be executed, and checks if the actual number of executions adheres to this constraint.

CCaaS is part of the *POD tool suite* NICTA has developed over the past two years,[5] where POD stands for Process-Oriented Dependability.

2 CCaaS Overview

In this section we discuss the core features of CCaaS, shown in Fig. 1: retrieving events and mapping them to process models, basic conformance checking, timing anomalies, and numerical invariants. Finally, we discuss the implementation.

2.1 Retrieving and Parsing Event Data

To retrieve log events in a timely fashion, a log agent can be deployed on the log-emitting system. We use the open-source tool Logstash[6] for this purpose. Since CCaaS is implemented as a RESTful service, it suffices to configure the Logstash agent to watch the location where logs are emitted (log file, database, or similar) and to forward each new log event to CCaaS via a HTTP invocation.

Once a new event is retrieved by CCaaS, we parse the event for the information that allows us to associate the event with a process model, instance, and activity in the model. This is log-type specific, and implemented with regular expressions. Relevant regular expressions can largely be learned during process discovery, e.g., using POD-Discovery [9], another tool from the POD tool suite.

2.2 Basic Conformance Checking

The main methods to check conformance are token replay [1], constraint checking [10, 5], and alignment [2]. In this work, we utilize token replay due to its conceptual integration with BPMN.

[5] `http://reliableops.com`
[6] `http://www.logstash.net/`, accessed on 6/6/2015

Token replay reenacts a historic log on the model by moving tokens through the model based on the observed events (after they have been mapped to activities in the model). If an activity's execution is observerd, but there is no token activating it, an error is thrown indicating the missing token, and a new token is inserted to activate the activity.

CCaaS accepts process models in BPMN. BPMN's execution semantics can be expressed as a token game, see e.g. [8]: tokens are placed on edges and an activity is activated if a token is present on its incoming edge, among others.

However, there is one challenge in implementing online token replay: for decision points (like XOR splits) it is a priori unclear which subsequent branch should be activated, i.e., which outgoing edge should receive a token. In essence, for the purposes of token replay we have to interpret each XOR split as a deferred choice. While in the process model the decision may be based on deterministic criteria, for CCaaS as an outside observer, the criteria are in general not visible. Hence we have to wait for the next observable event, i.e, activity execution, before we know which decision was made.

We solve the challenge using a *token pull mechanism*, which, in brief, works as follows. If activity A is observed, but not activated, we try to pull a token from its predecessor in the model. If that predecessor is another activity or a join gateway, the pull fails. If the predecessor is a split gateway, and is activated, the gateway is executed and the pull succeeds; if the split gateway is not activated, it attempts to pull a token from its predecessor, using the same logic. Multi-instantiation can be handled by CCaaS as described in [7].

2.3 Timing Anomalies

When business processes and activities are enacted multiple times and we measure the time spent in activities, we can learn the expected performance characteristics of activities. That is, we can learn the *duration distributions* of activities, which give us detailed insights into the timing behavior. We are interested in deviations from the expected behavior, e.g., if an activity is finished too early.

Based on the work of Rogge-Solti and Kasneci [6], we compute an activity's temporal anomaly intervals that correspond to unexpected durations. Therefore, we select a threshold of interest. For instance, we can set a threshold of 0.05, if we want to compute the intervals of the 5 percent most extreme cases.

Technically, we compute the boundaries of the intervals by looking at the distribution of the log-likelihood of random samples from the duration distribution as suggested by Yeung and Chow [12]. For example, with a 0.05 threshold, the cutoff value can be estimated by the 5th percentile of the log-likelihood distribution. The samples to the left of that value have lower log-likelihoods, and would be considered as outliers.

With the 5th percentile estimated, we use its value d of the probability density function (pdf) as a threshold. Finally, we subtract the density d from the pdf and need to find the roots of this shifted function. That is, we look for the regions where this shifted pdf is negative. In the case of non-parametric kernel density estimates using Gaussian heaps, we lack a closed analytic formula for the density.

Therefore, we apply a numeric root-finding algorithm as presented by Ford [3] to find the interval boundaries and export the interval regions.

Once timing anomaly intervals are computed for the activities, CCaaS compares the durations of the activities, as observed through the respective events, to the corresponding intervals. If an observed duration is anomalous, i.e., it lies within the anomaly intervals, an according error is raised to warn the analyst.

2.4 Numerical Invariants

In processes with iteration or multi-instantiation, some log events may contain numerical information about the number of iterations or multi-instantiated (MI) subprocesses that is expected if the current process instance operates correctly. For instance, in logs from Netflix Asgard[7] rolling upgrade, one log line states `"The group gr01 has 8 instances. 8 will be replaced, 1 at a time."` Subsequently, a loop is executed exactly 8 times, unless there is an error. One technique to discover such numerical invariants in logs can be found in the literature [4]. Assuming the discovery has taken place, a numerical invariant in CCaaS is defined by the following attributes:

- Start trigger event and position of the current values x_{min}, x_{max} in the event log, the lower and upper bounds, respectively. If there is only one exact value x, e.g., $x = 8$ in the example above, then $x_{min} := x =: x_{max}$.
- Scope of invariant effect, e.g., content of a loop or an MI subprocess.
- End trigger event, i.e., when all executions of the scope have to be completed.

Based on such a specification, CCaaS watches for the start trigger; once observed, the concrete values for x_{min} and x_{max} in the current process instance are known. Then, CCaaS tracks every start of the invariant's scope and retains the number x_{curr}. If $x_{curr} > x_{max}$ is observed, CCaaS raises an error. Once the end trigger is observed, CCaaS checks that $x_{min} \leq x_{curr} \leq x_{max}$, and that each of the x_{curr} instances of the scope completed successfully. Should any of these criteria not be met, an error is raised.

2.5 Implementation

CCaaS is implemented in about 1800 lines of Java code, using the Spring framework, and is available as open-source.[8] When invoked through the loopback network interface, average response time is typically below 10ms. In most deployments, higher network delays should be taken into consideration.

The computation and export of anomaly intervals are implemented in the `ProM` framework[9], also available as open-source. Starting from an enriched stochastic Petri net, the plugin can compute anomaly intervals for all transitions, which are then exported as JSON for online anomaly detection in CCaaS.

[7] `https://github.com/Netflix/asgard` accessed on 10/6/2015
[8] `https://github.com/NICTA/pod-detection/tree/master/cfmchecker`
[9] See the `StochasticPetriNet` package at `http://www.promtools.org`

3 Summary

General-purpose online conformance checking can support many usage scenarios. Any scenario where the *timeliness* of detecting non-conformance or errors is important can benefit strongly, e.g., so that the underlying issue can be corrected before it turns into a serious problem. Behavioral log analysis and tracking can be of great value to system reliability, as outlined in our previous work [9, 11]. The added capabilities for *timing anomalies* and *numerical invariants* bring this application to a new level: multiple concurrent subprocess instances can be tracked individually, in terms of fitness and the exact timing of their activities; we can ensure that the number of subprocess instances is exactly as expected and that each subprocess completes successfully. CCaaS has been developed over the last 18 months, and will be applied as part of the POD tool suite in trials with NICTA's business teams and industry partners in the near future.

Two screencast videos accompany this paper. First, we show CCaaS through the POD-Viz frontend.[10] Second, a brief screencast video shows the export of timing anomaly intervals from ProM.[11]

References

1. van der Aalst, W.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. van der Aalst, W., Adriansyah, A., van Dongen, B.: Replaying history on process models for conformance checking and performance analysis. WIREs Data Mining and Knowledge Discovery 2(2), 182–192 (2012)
3. Ford, J.A.: Improved algorithms of illinois-type for the numerical solution of non-linear equations. Tech. rep., University of Essex, Computer Science Dept. (1995)
4. Lou, J.G., Fu, Q., Yang, S., Xu, Y., Li, J.: Mining invariants from console logs for system problem detection. In: USENIX ATC (2010)
5. Maggi, F., Montali, M., Westergaard, M., van der Aalst, W.: Monitoring business constraints with linear temporal logic: An approach based on colored automata. In: BPM (2011)
6. Rogge-Solti, A., Kasneci, G.: Temporal anomaly detection in business processes. In: BPM (2014)
7. Weber, I., Farshchi, M., Mendling, J., Schneider, J.G.: Mining processes with multi-instantiation. In: ACM SAC (2015)
8. Weber, I., Hoffmann, J., Mendling, J.: Beyond soundness: On the semantic consistency of executable process models. In: IEEE ECOWS (2008)
9. Weber, I., Li, C., Bass, L., Xu, X., Zhu, L.: Discovering and visualizing operations processes with POD-Discovery and POD-Viz. In: IEEE/IFIP DSN (2015)
10. Weidlich, M., Polyvyanyy, A., Desai, N., Mendling, J., Weske, M.: Process compliance analysis based on behavioural profiles. Inf. Syst. 36(7), 1009–1025 (2011)
11. Xu, X., Zhu, L., Weber, I., Bass, L., Sun, W.: POD-Diagnosis: Error diagnosis of sporadic operations on cloud applications. In: IEEE/IFIP DSN (2014)
12. Yeung, D.Y., Chow, C.: Parzen-Window Network Intrusion Detectors. In: IEEE ICPR. vol. 4 (2002)

[10] `https://youtu.be/I-EjJGbvmzQ`

[11] `http://andreas.solti.de/temporal-anomaly-intervals`