Viola

Detecting Violations of Behaviors from Streams of Sensor Data

Di Federico, Gemma; Meroni, Giovanni; Burattin, Andrea

Published in:
Business Process Management Workshops - BPM 2023 International Workshops

Link to article, DOI:
10.1007/978-3-031-50974-2_10

Publication date:
2024

Document Version
Early version, also known as pre-print

VIOLA: Detecting VIOLAtions of Behaviors from Streams of Sensor Data

Gemma Di Federico[0000−0002−2487−1164], Giovanni Meroni[0000−0002−9551−1860], and Andrea Burattin[0000−0002−0837−0183]

Technical University of Denmark, Kgs. Lyngby, Denmark
gdfe@dtu.dk

Abstract. Sensor networks and the Internet of Things enable the easy collection of environmental data. With this data it is possible to perceive the activities carried out in an environment. For example, in healthcare, sensor data could be used to identify and monitor the daily routine of people with dementia. In fact, changes in routines could be a symptom of the worsening of the disease. Streaming conformance checking techniques aim at identifying in real-time, from a stream of events, whether the observed behavior differs from the expected one. However, they require a stream of activities, not sensor data. The artifact-driven process monitoring approach combines the structure of the control-flow with the data in an E-GSM model. This paper presents VIOLA, the first technique capable of automatically mining an E-GSM model from a labeled sensor data log, which is then suitable for runtime monitoring from an unlabeled sensor stream to accomplish our goal (i.e., streaming conformance checking). This approach is implemented and has been validated with synthetic sensor data and a real-world example.

1 Introduction

The environment we live in is becoming smarter and interconnected thanks to the increasing use of sensor systems. These systems are able to collect large amount of data on the environment we are living in and could be used to perceive the activities performed in order to derive their dependencies. In particular, these data can be represented in the form of a process model, where the process is intended as the execution of a series of activities performed in the environment.

One field, among many, that would benefit from this technology is the healthcare domain. In the area of Ambient Assisted Living, sensor systems can be used to discover and monitor the course of a disease. For instance, people affected by dementia usually tend to follow a strict routine. A deviation in the routine could be a symptom of the worsening of the disease. Therefore, by automatically modeling a person's routines as a process model [5], it becomes possible to monitor whether deviations from daily habits are occurring, in order to monitor the evolution of their disease.

However, data captured by sensor systems and behavioral process models are at two different levels of abstraction. Sensor data represent low-level events at

a specific point in time, whereas behavioral process models represent high-level activities and their dependencies. In addition, the application scenario requires real-time analysis so that medical staff can be notified as soon as changes are detected and react accordingly. This makes post-mortem analysis trivial. Therefore, discovering a behavioral process model from sensor data, detecting in real-time when activities in the model are executed, and identifying deviations between the discovered model and the observed behavior as they occur becomes a challenging task.

To address this issue, streaming conformance checking [2,15] aims to analyze an event stream with respect to a process model. However, most approaches for conformance checking focus on the structure of the process (i.e. the control flow), ignoring the data generated when the process is run (e.g. sensor data). Hence, they expect the stream to contain high-level events, rather than low-level sensor data. Therefore, a pre-processing step in charge of abstracting activities from sensor data would be necessary. Artifact-driven process monitoring is one of the few approaches that explicitly combines the structure of a process with the data produced by the activities in that process [10]. An example is the Extended-GSM (E-GSM) modeling language, which includes both control flow dependencies and rules to determine when activities are executed. However, artifact-driven process monitoring requires the process and the rules to identify activities to be manually modeled, either from scratch or from an imperative process model [11].

The work presented in this paper is called VIOLA, and it aims at detecting violations of behaviors from a stream of sensor data. The approach relies on Process Mining and Machine Learning techniques to automatically generate an E-GSM model from a labeled sensor log, where the labels indicate high-level activities carried out and perceived (i.e., observed) by the sensors. The streaming conformance checking algorithm takes as input directly the stream of raw sensors measurements, which lies at a lower abstraction level. The paper aims to answer the following research questions. First, we would like to focus on the derivation of the E-GSM process model, verifying if the approach is able to construct a behavioral process model suitable for conformance checking, where the data are in the form of labeled activities (**RQ1**). The resulting model is then used by the engine to monitor the process at run-time. In particular, the engine receives a stream of unlabeled sensor data and uses the E-GSM model to detect which activities are running. The second research question focuses on recognition, so we want to assess the approach's ability to recognize activities from an unlabeled stream of sensor data (**RQ2**). Once the running activities have been identified, the engine verifies the conformance between the observed behavior and its representation in the model. Therefore, the approach should be capable of verifying if the observed behavior is consistent with the discovered behavioral model (**RQ3**). The approach has been validated with an artificial sensor log and a real-world one.

The rest of the paper is structured as follows. Sect. 2 presents the related work. The approach is explained in Sect. 3. It is then evaluated in Sect. 4, including a discussion of the results. Sect. 5 concludes the paper.

2 Related Work

The level of abstraction at which a process model is represented is reflected in the granularity of the event log used to derive the model. When the process is captured by sensor systems, the data is in the form of fine-grained sensor measurements, which result in a chaotic and non-expressive process model [14]. Instead, experts are interested in analyzing the behavior in terms of higher-level activities. Hence, sensor measurements need to be grouped into activity labels.

Several techniques to cope with this problem have been devised, ranging from event abstraction [16] to activity recognition [1]. One of the main challenges of this work is to recognize activities on a stream of data, i.e. online. In the field of event abstraction, Tax et al. [13] propose a supervised learning approach that applies a statistical modeling method to recognize activities. The approach takes as input a set of annotated traces used to train a conditional random field, which then operates on an unlabeled log to abstract activities. Moving to the activity recognition field, Sanchez et al. [12] propose a technique to recognize abnormal activities. They use a labeled event log to train a classifier, which is then tested on an unlabeled log. The approach is only capable to detect activities that were already included in the training set. Also Maswadi et al. [9] make use of classification algorithms, comparing Decision Tree (DT) and Naïve Bayes classifiers with the aim to perform a real-time classification of activities. The performance of the two algorithms is similar, but the DT classifier resulted in higher accuracy. The authors also highlight the importance of data pre-processing for feature selection, which can significantly improve classification accuracy. What is more, the DT is more clear in the explanation and representation of the decisions.

Furthermore, the process model must be able to represent activities but also consider the information behind them. That is, make the process data-aware. For the imperative languages, the focus is on data-aware Petri Nets [7, 8]. E.g., the Colored Petri Net, where the data dimension is dependent on the control flow as data elements are attached to the tokens. During the verification of a Colored Petri Net [4], both the data and the control flow are checked to evaluate the next activity. When a violation is identified, the verification of the process instance is stopped. Moving towards declarative languages, Burattin et al. [3] recognize the need of verifying the conformance of multi-perspective declarative models, by proposing a conformance checking algorithm that exploits also the data dimension. It is worth mentioning that all of the aforementioned approaches revolve around the notion of *activity*, and the data is seen as an event attribute. In the context of this paper, the objective is to deal directly with a stream of data values, where the notion of activity naturally emerges from the data configuration observed. On the one hand, all the mentioned approaches do not allow for the streaming verification of the process. On the other hand, the existing online conformance checking algorithms [2, 15] only focus on the control flow without considering the data. A solution is presented by Meroni et al. [11], E-GSM, in which both the data and the process perspectives are considered, by introducing the so-called process flow guards. The approach allows for the run-time verification of a process. However, no approach to discover E-GSM

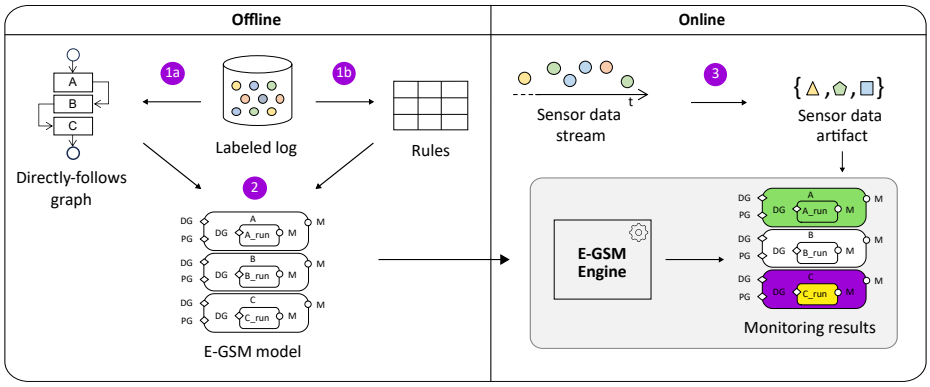


Fig. 1: Overview of the VIOLA approach. The different steps of the approach are numbered 1a, 1b, 2, and 3

models from execution logs exists. Also, to determine when activities are executed, the E-GSM model predicates on labels representing the discrete state of the environment, rather than on raw sensor data.

3 Approach

The approach presented in this paper, named VIOLA, aims to identify violations of behaviors from a stream of sensor data. The streaming conformance checking, based on an E-GSM model, is able to process a stream of sensors, represent it in the form of an abstracted process model, and verify at run-time if new instances of the process are compliant.

An overview of the VIOLA approach is presented in Figure 1. VIOLA is divided into two phases, one offline and one online. During the offline phase, the E-GSM model is constructed. The starting point is a labeled sensor log.*Step 1a* consists of the derivation of the process flow conditions, in the form of a Directly-Follows Graph (DFG). In *Step 1b*, the data flow constraints are derived and represented in the form of rules. The rules explain how each activity label can be recognized, starting from an unlabeled series of sensor measurements. In *step 2*, both the DFG and the rules are used to build the E-GSM model. In particular, information in the DFG is used to derive the structure of the process, whereas the rules are used to derive the conditions on the *data flow guards* and *milestones*. In the online (i.e., streaming) phase, a stream of unlabeled sensor data is processed (*step 3* of Fig. 1). The processing consists in dividing the stream in windows to be analyzed and organizing them in the sensor data artifact suitable for the E-GSM engine. Based on the content of the windows and the constraints on the stages, running activities are recognized. If the rules on the guards are not fulfilled, violations in the execution of the process are identified.

In the following sections, each step of the two phases is explained in detail. But before, background knowledge on E-GSM is provided.

3.1 Background

E-GSM is an extension of the Guard-Stage-Milestone (GSM) artifact-centric language explicitly designed for process monitoring. As shown in Figure 3, E-GSM represents the units of work performed in a process (e.g., activities and process portions) with stages. Stages can be either atomic (e.g., A_run), or can nest other stages (e.g., A).

To determine when a stage is executed, it is annotated with data flow guards (e.g. $A.DG$ and $A\_run.DG$), and milestones (e.g., $A.M$ and $A\_run.M$). Data flow guards and milestones are Event-Condition-Action (ECA) rules, which require an event and a boolean expression to be specified. In particular, the boolean expression in an ECA rule is evaluated when the specified event happens, based on the state of the model at that time.

Stages can be in one of the following states: *unopened*, *opened*, *closed*. When the process starts, all stages are *unopened*, indicating that they were never executed. When a data flow guard holds and the associated stage is *unopened* or *closed*, that stage becomes *opened*, indicating that the process element represented by that stage is being executed. When a milestone holds and the associated stage is *opened*, that stage becomes *closed*, indicating that the process element completed its execution. Also, if that stage has any *opened* child stages, those become *closed* too.

A stage can be decorated with process flow guard (e.g., $A.PG$) to represent control flow dependencies that a stage should fulfill before being executed. A process flow guard is a boolean expression that is evaluated immediately after one of the data flow guards of the associated stage holds.

Stages can also be marked as *onTime*, *outOfOrder*, or *skipped*. When the process starts, all stages are *onTime*. When one of the data flow guards of a stage holds, the expression on the process flow guard of that stage is evaluated. If the expression evaluates to `true`, the associated stage remains *onTime*, *outOfOrder* otherwise. In addition, if the expression predicates on another stage being *opened* or *closed*, and that stage is *unopened*, then that stage becomes *skipped*.

It is worth noting that process flow guards do not enforce control flow dependencies. A stage can still be opened even if its process flow guard does not hold, as long as at least one of its data flow guards holds. This allows to continuously and autonomously monitor the execution of a process based on sensor data.

E-GSM supports different kinds of events. In this paper, we will cover only a subset that is relevant for VIOLA. When the E-GSM engine receives updated sensor data, it triggers two events e' and e'' to be emitted in succession. These events can then be used by data flow guards and milestones (e.g., $A.DG$ and $A\_run.M$) to check their boolean expression only when the sensor data changes. Also, when a milestone holds, it triggers an event (e.g., $A\_run.M$ when the milestone M of A_run is achieved) to be emitted. Such event can then be used by data flow guards and milestones (e.g., $A.M$) to check their value only when a portion of the process completed its execution.

Boolean expressions can predicate both on sensor data to check whether they assume a specific value (e.g., $A.DG$ and $A\_run.M$) and on the model itself to check whether a stage is *opened* (e.g., $A.PG$ and $A.M$).

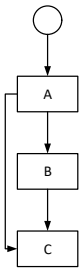
3.2 Steps 1a and 1b: Derivation of DFG and Decision Rules

The key aspect of the proposed approach is its capability of processing an unlabeled stream of data. To achieve the objective, an E-GSM model must be derived. For the automatic generation of the model, information related to the control flow and to the activities must be provided. As depicted in Figure 1, the first phase is composed of two steps. In *step 1a* the control flow is discovered by means of a Process Mining algorithm, that derives a DFG. In the DFG, the nodes represent activities while the edges are the directly follows relationships between pairs of activities. We also assume that it is possible to augment the DFG with information regarding which nodes are starting nodes. This information can be extracted either from the DFG itself (e.g., by looking for nodes with no incoming edges) or by looking for activities that are predominantly observed as the first activity in the log (the actual procedure to perform this is outside the scope of this paper). The DFG is used to define, in the E-GSM model, the stages and their process flow guards. Since the E-GSM model is derived starting from the DFG, the expressiveness of the E-GSM model is fundamentally limited by the expressiveness of DFG. In particular, parallel executions are not allowed.

The stage names on the E-GSM model have the label of the high-level activities from the labeled sensor log. However, during the streaming conformance checking phase, an unlabeled stream is processed. For this reason, we have to derive the rules necessary to recognize an activity (*step 1b*), that will be used to define the data flow guards and milestones in the E-GSM model. We require that the same activity label cannot be repeated in a sequence, since ideally the same activity is not performed several times in a row. What is more, we assume that there is no interleaving in the execution of activities. An activity is characterized by a specific series of sensor data, and this information is used to train a classifier. For example, we can segment the sensor log into windows and, for each window, determine the set of sensors required to detect the corresponding activity label. Sensor names could then be used as features to train a DT classifier, where the leaves (i.e., the classes) refer to the activity labels, while the internal nodes predicate on the selected classification features. The classifier is translated into first-order logic expressions, each referring to the data conditions for a certain activity to occur.

3.3 Step 2: Generation of E-GSM Models from DFGs and Decision Rules

Once we have discovered the DFG and we have identified the rules to infer from sensor data when an activity is running, we can build an E-GSM model (*step 2* in Fig. 1) that will be used to monitor the process at runtime (i.e., streaming conformance checking).



(a) DFG

Label	Rule
A	x>1
B	y>2
C	z>1

(b) Rules

Fig. 2: DFG (a) and rules (b) extracted from labeled sensor log

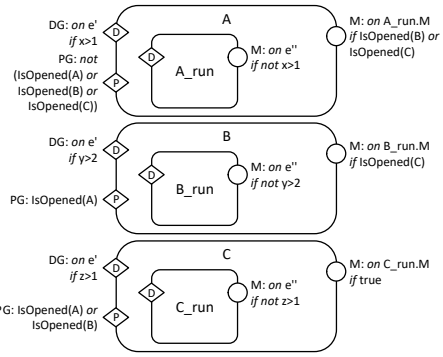


Fig. 3: Generated E-GSM model

Since an activity in a DFG can be connected with any other activity, existing techniques to generate E-GSM models from block-structured process models are not suited for this purpose. Also, approaches that enforce control flow dependencies cannot be used, since they would not allow monitoring executions that differ from the expected control flow. Therefore, we defined an entirely new transformation logic to derive the E-GSM model.

In particular, for each activity T in the DFG, the following elements are derived in the E-GSM model.

- **One atomic stage** S_run, which is used to detect when T is running.

  For example, stages A_run, B_run and C_run in Figure 3 are derived from, respectively activities A, B and C in Figure 2a.

- **One parent stage** S, which contains S_run and it is used to assess the conformance of T with respect to the expected execution flow.

  For example, stages A, B and C in Figure 3 are derived from, respectively activities A, B and C in Figure 2a.

- **One data flow guard** $S.DG$, which is attached to S. $S.DG$ is evaluated when event e' is emitted (i.e., every time updated sensor data are received, before e'' is emitted). $S.DG$ requires the rules from the decision tree associated with the class T to evaluate to `true`. In this way, whenever updated sensor data are received, $S.DG$ is evaluated. If the sensor data indicate that activity T is being executed (that is, $S.DG$ evaluates to `true`), stage S will become *opened*.

  For example, the data flow guard $A.DG$ in Figure 3 requires the rule in Fig. 2b associated with activity A, that is, the value registered by sensor x to be greater than 1, to evaluate to `true`.

- **One data flow guard** $S\_run.DG$, which is attached to S_run, which is identical to $S.DG$.

In this way, when activity A is being executed, stage A_run will be *opened*.

- **One milestone** $S\_run.M$, which is attached to S_run. $S\_run.M$ is evaluated when event e'' is emitted (i.e., every time updated sensor data are received, immediately after e' is emitted). $S\_run.M$ requires the rule associated to T to evaluate to `false`. In this way, whenever updated sensor data are received, $S.M$ is evaluated. If the sensor data indicate that activity T is no longer being executed (that is, $S.DG$ evaluates to `false`), stage S will become *closed*. Also, since e'' is always emitted after e', whenever updated sensor data are received, $S\_run.M$ will always be evaluated after all the data flow guards in the model.

  For example, the milestone $A\_run.M$ in Figure 3 requires the fulfillment of the rule in Fig. 2b associated with activity A, that is, the value registered by sensor x to be greater than 1, to evaluate to `false`.

- **One milestone** $S.M$, which is attached to S. $S.M$ is evaluated when $S\_run.M$ is emitted. $S\_run.M$ requires at least one stage S_{succ} to be *opened*. S_{succ} is a parent stage derived from an activity T_{succ} which is a successor of T, that is, in the DFG T_{succ} is directly connected to T through an incoming arc. If no stage S_{succ} exists, then $S.M$ always evaluates to `true`.

  For example, the milestone $A.M$ in Figure 3 is evaluated when $A\_run.M$ is emitted, and requires either stage B or C, which are derived from the activities in Figure 2a that are successors of A, to be *opened*. In this way, when activity A stops being executed, stage A will become *closed* only if the next activity is either B or C.

- **One process flow guard** $S.PG$, which is attached to S. $S.PG$ requires at least one of the following boolean expressions to evaluate to `true`:
  1. At least one stage S_{pred} must be *opened*. S_{pred} is a parent stage derived from an activity T_{pred} which is a predecessor of T, that is, in the DFG T_{pred} is directly connected to T through an outgoing arc. If no stage S_{pred} exists, then this expression always evaluates to `false`.
  2. If T is an initial activity, that is, in the DFG the process start is directly connected to T with an outgoing arc, no parent stage in the model must be *opened*. If T is not an initial activity, then this expression always evaluates to `false`.

  For example, the process flow guard $A.PG$ in Figure 3 requires that stages A, B and C must not be *opened*, since activity A in Figure 2a is an initial activity and has no predecessor. Conversely, the process flow guard $B.PG$ in Figure 3 requires that stage A must be *opened*, since activity A in Figure 2a is a predecessor of activity B.

3.4 Step 3: Streaming Processing and Conformance Checking

The last step of the approach, *step 3* in Figure 1, consists of the conformance checking of a stream of sensor data. During the online processing, the stream of

sensor data passes through a pre-processing phase where the data are converted into the format suitable for the E-GSM engine. As introduced in Sect. 3.2 for the *step 1b*, the sensor data stream can be processed in windows. During the segmentation, we determine the set of sensors contained in each window, which is transformed into a data object (i.e. the sensor data artifact), and sent to the E-GSM engine. Each data object is evaluated by the engine according to the data flow guards and milestones, in order to check whether the target conditions are met, that is, if the activity is recognized and running.

To conclude, the approach presented in this paper allows the discovery of a E-GSM model starting from a labeled sensor log. The model is then suitable for the streaming conformance checking of an unlabeled sensor stream, in order to detect deviations in the behaviors.

4 Evaluation

To evaluate the approach presented in this paper, two experiments are conducted, one based on synthetic event logs, and the other based on a real dataset. The objective of the evaluation is to verify whether the approach is able to answer the research questions presented in Sect. 1. In particular, the use of synthetic event logs allowed us to conduct a controlled experiment where we could scrupulously verify the approach's ability to process a stream of events, correctly recognize activities, and verify the conformity of the behavioral process model. The experiment using a real dataset served to verify the actual applicability of the approach to a real use case. In the context of this paper, only the first experiment is explained, while the second one can be found in a technical report[1].

The first experiment is based on two synthetic datasets we constructed[2]. The datasets describe the movements of a person inside a smart environment, where movements are captured by sensors. To obtain the datasets, we used a smart environment simulator tool, called Linac [6], which produces as output a stream of triggered sensors, generated by the movements of a previously programmed agent. The stream has been sequentialized and stored in a single log file and it has been processed in order to be labeled. Two scenarios are constructed: a normal scenario used to discover the process model and construct the E-GSM, and a variation (i.e. a misalignment) used in the online phase for streaming conformance checking.

The map of the environment used in the simulation is composed of four rooms (named *A, B, C, D*). Each room is equipped with 5 pressure sensors on the floor that record pressure variations, and a presence sensor on the entry door that triggers at the passage in the detecting area. Sensors have a trigger frequency of 30 seconds. The person moves across the rooms, and the behavior changes according to the situation being simulated. In the base scenario (named l-base), the person moves between the four rooms, following the order from *A*

[1] The appendix can be found at https://dx.doi.org/10.5281/zenodo.7982452.
[2] The source code can be found at https://github.com/gemmadifederico/VIOLA.

	l-base	l-error
False positives	0	8
False negatives	0	1
True positives	400	235
Precision	1	0.97
Recall	1	0.99
F1 score	1	0.98

(a) Activity recognition evaluation

	l-base	l-error
onTime activities	400	50
outOfOrder act.	0	185
Tot. activities	400	236
Conformance	1	0.21

(b) Control flow evaluation

Table 1: Results of the synthetic simulation

to *D*. The log constitutes the base case, and it is used to verify the capability of the approach in detecting activities. In the variation of the scenario (named *l-error*), the path of the person varies, covering either all the rooms in a different order or just some of them. The variation is in the control flow, and the goal is to recognize the set of activities executed but also identify when the set differs from the modeled one, i.e. the violations.

The base behavioral model is derived from the normal scenario. The l-base log is split into training and test sets (70/30). Following the procedure indicated in Fig. 1, a DFG and a set of rules are derived. In our tests, we used the PM4Py and Scikit-learn libraries for Python[3] for deriving, respectively, the DFG and the set of rules. Firstly, the training set is pre-processed to group events into timed windows of 30 seconds. The size of the window was decided based on the average duration of each activity identifier, trying to approximate it. For each window, we extract the set of sensors and the corresponding activity label. The sensor names are used as features to train a classifier. The CART (Classification and Regression Tree) algorithm is used for the classification, while the Gini Index is adopted for the splitting. The obtained DT is then translated into boolean expressions to obtain the rules. Secondly, the corresponding function of the aforementioned library is used to derive a DFG representing the control flow of the process. Therefore, the DFG and the rules are translated in the E-GSM behavioral model, which is passed to the engine for the online phase. SMARTifact [10] is the process monitoring platform used by the engine. During the online phase, the logs are pre-processed by a Complex Event Processing (CEP) system, which simulates a stream, hides the label, and groups the data in timed windows (as in offline processing).

The results of the application of the approach are presented in Table 1. The evaluation covers the recognition of activities considering false positives, false negatives, and true positives (Table 1a); while the control flow is evaluated by considering onTime and outOfOrder activities, as well as the conformance measure (Table 1b). The first columns in Tables 1 refer to the verification with the test set of the normal scenario. As can be noticed in Table 1a, all the activities

[3] See https://pm4py.fit.fraunhofer.de/, https://scikit-learn.org/stable/.

are correctly recognized, returning a perfect value for precision, recall, and F1 score. When the control flow varies, that is l-error, it becomes more challenging to recognize the activities. However, the approach was able to recognize almost the entire set of activities (indeed precision and recall values are still high), starting from the sensor stream. Once the activities have been recognized, we need to verify if they conform to the control flow. An activity identified among the true positives is flagged as *onTime* when it fulfills the requirements of both the data and the process flow guards. If an activity is detected, and the data flow guard holds but the process flow guard doesn't hold, the activity is marked as *outOfOrder*. Therefore, the overall conformance is computed as the ratio between the total amount of *onTime* activities and the sum between the total *onTime* and the total *outOfOrder*. The results are shown in Table 1b. In the verification using the test set of the normal scenario, the control flow perfectly fits the reference model, as expected. For the variation instead, it is important to consider that not all the activities in l-error are non-compliant. In fact, there could be a partial match of the control flow. As a consequence, not all the activities are marked as *outOfOrder*.

The application of the approach on synthetic datasets produced promising results as the discovered data guards are capable of discriminating the different activities. Hence, we are able to construct a correct and meaningful behavioral model suitable for conformance checking. Therefore, we can positively answer to RQ1. In the base case, all the activities were correctly recognized. Also, no control flow violations were detected. Even with the introduction of variations in the behaviors, only a few activities were erroneously classified. Similarly, only a few non-existent control flow violations were detected. This aspect partially answers to RQ2 and RQ3, since it could generate a cascade effect. If an activity is mislabeled and therefore it is flagged as *running*, but violates the control flow, it will impact the conformance of the entire process instance. In other words: if an activity is wrongly classified, then the conformance will be evaluated against the classified activity, not the real one.

5 Conclusions and Future Work

In this paper, we presented VIOLA, a behavioral violations detection approach that works on streams of sensor data. VIOLA is able to automatically derive an E-GSM behavioral model starting from a labeled sensor log. What is more, VIOLA allows the online verification of the conformity between the behavioral process model and an unlabeled stream of sensor data. The behavioral model represents activity labels as stages, while the stages predicate on the sensor data. In other words, the approach is able to process a stream of raw sensor events (i.e. low-level of granularity), while the model represents higher-level activity labels. The approach is evaluated using synthetic and real datasets. In both cases positive results were obtained, supporting the quality of the proposed approach. The approach has some limitation mainly due to the expressiveness of the DFG graph used to generate the E-GSM model. The approach does not allow for

parallel executions. What is more, we do not consider repetitive activities, i.e., when the same activity is repeated several times in a sequence. In fact, we assume that a person does not perform the same activity over and over again, but only that they are performing that specific activity. As a future work, we would like to improve the detection of activities by enhancing the hyperparameter tuning process, as it influences the performance of the task. In addition, we aim to formalize the translation from the DFG and the rules in the E-GSM model, in order to demonstrate their correctness.

References

1. Arshad, M.H., Bilal, M., Gani, A.: Human activity recognition: Review, taxonomy and open challenges. Sensors **22**(17), 6463 (2022)
2. Burattin, A., Carmona, J.: A framework for online conformance checking. In: BPM. pp. 165–177. Springer (2017)
3. Burattin, A., Maggi, F.M., Sperduti, A.: Conformance checking based on multi-perspective declarative process models. Expert Syst. Appl. **65**, 194–211 (2016)
4. Carrasquel, J.C., Mecheraoui, K., Lomazova, I.A.: Checking conformance between colored petri nets and event logs. In: AIST. pp. 435–452. Springer (2020)
5. Di Federico, G., Burattin, A., Montali, M.: Human behavior as a process model: Which language to use? In: ITBPM@ BPM. pp. 18–25 (2021)
6. Di Federico, G., Nikolajsen, E.R., Azam, M., Burattin, A.: Linac: A smart environment simulator of human activities. In: ICPM. pp. 60–72. Springer (2022)
7. Felli, P., Gianola, A., Montali, M., Rivkin, A., Winkler, S.: Cocomot: Conformance checking of multi-perspective processes via smt. In: BPM. pp. 217–234 (2021)
8. de Leoni, M., Felli, P., Montali, M.: A holistic approach for soundness verification of decision-aware process models. In: ICCM. pp. 219–235. Springer (2018)
9. Maswadi, K., Ghani, N.A., Hamid, S., Rasheed, M.B.: Human activity classification using decision tree and naive bayes classifiers. Multimed. Tools. Appl. **80**(14), 21709–21726 (2021)
10. Meroni, G.: Artifact-Driven Business Process Monitoring - A Novel Approach to Transparently Monitor Business Processes, Supported by Methods, Tools, and Real-World Applications, Lect. Notes Bus. Inf., vol. 368. Springer (2019)
11. Meroni, G., Baresi, L., Montali, M., Plebani, P.: Multi-party business process compliance monitoring through iot-enabled artifacts. Inf. Syst. **73**, 61–78 (2018)
12. Sánchez, V.G., Skeie, N.O.: Decision trees for human activity recognition modelling in smart house environments. Simul. Notes Eur. **28**, 177–184 (2018)
13. Tax, N., Sidorova, N., Haakma, R., Aalst, W.: Mining process model descriptions of daily life through event abstraction. In: Int. J. Intell. Syst. Appl. pp. 83–104. Springer (2016)
14. Tax, N., Sidorova, N., Haakma, R., van der Aalst, W.M.: Log-based evaluation of label splits for process models. Procedia Comput. Sci. **96**, 63–72 (2016)
15. van Zelst, S.J., Bolt, A., Hassani, M., van Dongen, B.F., van der Aalst, W.M.: Online conformance checking: relating event streams to process models using prefix-alignments. JDSA **8**(3), 269–284 (2019)
16. van Zelst, S.J., Mannhardt, F., de Leoni, M., Koschmider, A.: Event abstraction in process mining: literature review and taxonomy. Granul. Comput. **6**(3) (2021)