Lights, Camera, Action! Business Process Movies for Online Process Discovery

Andrea Burattin[1], Marta Cimitile[2], and Fabrizio Maria Maggi[3]

[1] University of Padua, Italy
burattin@math.unipd.it
[2] Unitelma Sapienza University, Italy
marta.cimitile@unitelma.it
[3] University of Tartu, Estonia
f.m.maggi@ut.ee

Abstract. Nowadays, organizational information systems are able to collect high volumes of data in event logs every day. Through process mining techniques, it is possible to extract information from such logs to support organizations in checking process conformance, detecting bottlenecks, and carrying on performance analysis. However, to analyze such "big data" through process mining, events coming from process executions (in the form of event streams) must be processed on-the-fly as they occur. The work presented in this paper is built on top of a technique for the online discovery of declarative process models presented in our previous work. In particular, we introduce a tool providing a dynamic visualization of the models discovered over time showing, as a "process movie", the sequence of valid business rules at any point in time based on the information retrieved from an event stream. The effectiveness of the visualizer is validated through an event stream pertaining to health insurance claims handling in a travel agency.

Keywords: Online Process Discovery, Dynamic Process Model Visualization, Event Stream Analysis, Declarative Process Models, Concept Drifts, Operational Decision Support

1 Introduction

Process innovation and optimization is a crucial step of the business process management lifecycle of an organization. In addition, nowadays, high volumes of data are available and can be used for process improvement. In this scenario, process mining techniques [1] may support analysts in extracting useful information from data recorded during process executions in the so-called event logs that can provide valuable insights for process enhancement. In particular, the goal of one of the main branches of this discipline, process discovery, is to build process models representing the process behavior as recorded in the logs. However, one of the challenges that exist when several gigabytes of data need to be analyzed is to process the data on-the-fly through techniques that work at runtime and analyze the data as soon as it becomes available. Unfortunately, traditional process discovery techniques are limited to the off-line analysis of event logs and require several iterations on the data for discovering a process model. For this reason, these techniques cannot be used in an online setting with the consequence of a reduced capability to support real-time process comprehension and monitoring.

In [15], we propose an approach to automatically discover declarative process models from streams of data, at runtime. A declarative process model is a set of business rules that describe the process behavior under an open world assumption, i.e., everything that is not forbidden by the model is allowed. These models can be used to express process behaviors involving multiple alternatives in a compact way and are very suitable to be used in changeable and instable environments with respect to the conventional procedural approaches. In [15], an online process discovery technique is proposed to automatically construct, from an event stream, a representation of complex business processes in the form of business rules, without using any a-priori information and processing the events on-the-fly, as they occur. In addition, this approach is able to detect concept drifts, i.e., changes in the process execution due to periodic/seasonal phenomena that are vital to be detected and analyzed to deeply understand how the process adapts its behavior over longer periods of time.

Even if this approach has been implemented in the process mining tool ProM [19], its previous implementation was able to cope only with "static" objects and a graphical visualization of the evolution of the process model over time was missing. In this work, we try to address this limit by presenting a graphical visualizer for process models extracted from an event stream through the declarative process discovery approach presented in [15]. The visualizer has been developed using a client-server architecture in which the stream source is the server and a number of miners, modeled as clients, can be connected to the source using TCP connections. When a new event is emitted, a log fragment is encoded only containing that event and sent to the clients. The reconstruction of the complete final process instance (by grouping together events belonging to the same case) is delegated to the miners. The output models are visualized in the form of an animated "process movie" that dynamically shows how the behavior of the process that produces the event stream changes over time. The process models are represented using Declare [2] a declarative process modeling language to describe process behaviors through business rules. The added value provided by the visualizer to the process stakeholders is to have continuously updated "de facto" models showing the real behavior of the process as recorded in the stream.

The paper is structured as follows. Section 2 introduces the characteristics of event stream mining as well as some basic notions about Declare. Here, we also give a short description of the online process discovery technique underlying the proposed visualizer. Section 3 introduces the visualizer and describes its implementation. In Section 4, the tool is applied to an event stream pertaining to health insurance claims handling in a travel agency. Finally, Section 5 reports some conclusion and final remarks.

2 Preliminaries

This section provides a general introduction to the basic elements we are going to use throughout the paper. In particular, we introduce the notion of event stream and we give a quick overview of the Declare language. Also, we introduce the approach for the online discovery of Declare models that we use to produce the process movies presented in this paper.

2.1 Event Streams

Several works in the data mining literature, such as [12, 3, 4], agree in defining a *data stream* as a fast sequence of data items. It is common to assume that: *(i)* data has a small, typically predefined, number of attributes; *(ii)* mining algorithms are able to analyze an infinite number of data items, handling problems related to memory bounds; *(iii)* the amount of memory that the learner can use is finite and much smaller than the memory required to store the data observed in a reasonable span of time; *(iv)* each item is processed within a certain small amount of time (algorithms have to linearly scale with respect to the number of processed items): typically the algorithms work with one pass on the data; *(v)* data models associated to a stream (i.e., the "underlying concepts") can either be stationary or evolving [20].

An event stream is a potentially infinite sequence of events. It is possible to assume that sequence indexes comply with the time order of events (i.e., given the sequence S, for all indexes $i \in \mathbb{N}^+$, $\#_{\text{time}}(S(i)) \leq \#_{\text{time}}(S(i+1))$). Starting from a general data stream (not necessarily a stream of events), it is possible to perform different analysis. The most common are clustering, classification, frequency counting, time series analysis, and change diagnosis (concept drift detection) [10, 20, 3].

2.2 Declare: Some Basic Notions

In this paper, the process behavior as recorded in an event stream is described using Declare rules. Declare is a declarative process modeling language originally introduced by Pesic and van der Aalst in [2]. Instead of explicitly specifying the flow of the interactions among process events, Declare describes a set of rules that must be satisfied throughout the process execution. The possible orderings of events are implicitly specified by rules and anything that does not violate them is possible during execution. In comparison with procedural approaches that produce "closed" models, i.e., all what is not explicitly specified is forbidden, Declare models are "open" and tend to offer more flexibility for the execution.

A Declare model consists of a set of rules applied to events. Declare rules, in turn, are based on templates. Templates are patterns that define parameterized classes of properties, and Declare rules are their concrete instantiations. Templates have a user-friendly graphical representation understandable to the user and their semantics can be formalized using different logics [17], the main one

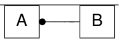
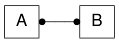
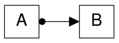
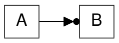
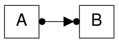
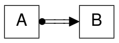
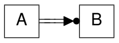
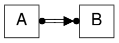
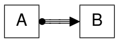
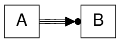
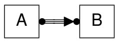
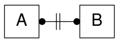
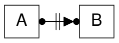
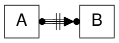
Template	Meaning	Notation
Responded Existence(A,B)	if A occurs then B occurs before or after A	
Co-Existence(A,B)	if A occurs then B occurs before or after A and vice versa	
Response(A,B)	if A occurs then eventually B occurs after A	
Precedence(A,B)	if B occurs then A occurs before B	
Succession(A,B)	for A and B both precedence and response hold	
Alternate Response(A,B)	if A occurs then eventually B occurs after A without another A in between	
Alternate Precedence(A,B)	if B occurs then A occurs before B without another B in between	
Alternate Succession(A,B)	for A and B both alternate precedence and alternate response hold	
Chain Response(A,B)	if A occurs then B occurs in the next position after A	
Chain Precedence(A,B)	if B occurs then A occurs in the next position before B	
Chain Succession(A,B)	for A and B both chain precedence and chain response hold	
Not Co-Existence(A,B)	A and B cannot occur together	
Not Succession(A,B)	if A occurs then B cannot eventually occur after A	
Not Chain Succession(A,B)	if A occurs then B cannot occur in the next position after A	

Table 1. Graphical notation and meaning of the Declare templates.

being LTL, making them verifiable and executable. Each rule inherits the graphical representation and semantics from its templates. The major benefit of using templates is that analysts do not have to be aware of the underlying logic-based formalization to understand the models. They work with the graphical representation of templates, while the underlying formulas remain hidden. Table 1 summarizes some of the Declare templates (we indicate template parameters with capital letters and concrete activities in their instantiations with lower case letters). For a complete overview on the language the reader is referred to [18].

Consider, for example, a *response* rule connecting activities a and b. This rule indicates that if a *occurs*, b must eventually *follow*. Therefore, this rule is satisfied for traces such as $\mathbf{t}_1 = \langle a, a, b, c \rangle$, $\mathbf{t}_2 = \langle b, b, c, d \rangle$, and $\mathbf{t}_3 = \langle a, b, c, b \rangle$, but not for $\mathbf{t}_4 = \langle a, b, a, c \rangle$ because, in this case, the second instance of a is not followed by a b. Note that, in $\mathbf{t}_2$, the considered response rule is satisfied in a trivial way because a never occurs. In this case, we say that the rule is *vacuously satisfied* [14]. In [6], the authors introduce the notion of *behavioral vacuity detection* according to which a rule is non-vacuously satisfied in a trace when it is activated in that trace. An *activation* of a rule in a trace is an event whose occurrence imposes, because of that rule, some obligations on other events in the same trace. For example, a is an activation for the *response* rule between a and b, because the execution of a forces b to be executed eventually.

An activation of a rule can be a *fulfillment* or a *violation* for that rule. When a trace is perfectly compliant with respect to a rule, every activation of the rule in the trace leads to a fulfillment. Consider, again, the response rule connecting activities a and b. In trace $\mathbf{t}_1$, the rule is activated and fulfilled twice, whereas, in trace $\mathbf{t}_3$, the same rule is activated and fulfilled only once. On the other hand, when a trace is not compliant with respect to a rule, an activation of the rule in the trace can lead to a fulfillment but also to a violation (at least one activation leads to a violation). In trace $\mathbf{t}_4$, for example, the response rule between a and b is activated twice, but the first activation leads to a fulfillment (eventually b occurs) and the second activation leads to a violation (b does not occur subsequently). An algorithm to discriminate between fulfillments and violations for a rule in a trace is presented in [6].

2.3 Online Discovery of Declare Models

In this section, we describe how to discover, at runtime, a Declare model out of a stream of events. The approach for online discovery of Declare models we use in this paper is based on three algorithms. The first algorithm is called *Sliding Window* [11]. The basic idea of this algorithm is to keep the latest observed events and consider them as a static event log. The other two approaches are called *Lossy Counting* [16] and *Lossy Counting with Budget* [9]. The basic idea of these approaches is to consider aggregated representations of the latest observations, i.e., instead of storing repetitions of the same event, to save space, they store a counter keeping trace of the number of instances of the same observation. As presented in [15], we use these algorithms for memory management in combination with algorithms for the online discovery of Declare rules referring to different templates.

Algorithm 1: Online discovery scheme

```
Input: $S$: event stream, *conf*: approximate algorithm configuration (either LC or LCB, with
       the corresponding configuration: $\epsilon$ or $\mathcal{B}$), *update model condition*
1  $R \leftarrow \emptyset$                                   /* Set of template replayers */
2  foreach Declare template $t$ do
3      Add a replayer for $t$ (according to conf) to $R$
4  end
5  forever do
6      $e \leftarrow observe(S)$
7      if $analyze(e)$ then
8          foreach $r \in R$ do
9              $r(e, conf)$                            /* Replay $e$ on replayer $r$ */
10         end
11     end
12     if update model condition then
13         model $\leftarrow$ initially empty Declare model
14         foreach $r \in R$ do
15             Update model with rules in $r$
16         end
17         Use model              /* For example, update a graphical representation */
18     end
19 end
```

Algorithm 1 presents a general overview of our online discovery approach. Here, we keep a set R of *replayers*, one for each template we want to discover (line 3). Note that replayers for different templates implement different discovery algorithms each one developed based on the characteristics of the corresponding template. Then, when a new event e is observed from the stream S, if it is necessary to analyze it, the algorithm replays e on all the template replayers of R (line 9). Periodically (the period may depend either on the number of events observed, or on the actual elapsed time), it is possible to update the discovered Declare model, by querying each replayer for the set of satisfied rules (line 15). These models are used in our visualizer as "frames" to build the process movies.

The candidate rules to be included in the discovered Declare model can be selected based on the percentage of activations that leads to a fulfillment (event-based rule support) or based on the percentage of cases in which the rule is satisfied (trace-based support). Of course, in this latter case, the event stream should provide not only information about the case to which each event belongs but also an indication on the exact point in which each trace starts and ends.

3 The Visualizer

The visualizer described in this paper has been implemented as plug-in of the process mining tool ProM.[4] In the following sections we describe the tool and its characteristics.

3.1 The Event Streamer

The current implementation of ProM is able to cope only with "static" objects, e.g., log files and process models. Instead, as previously stated, an event stream cannot be fit into a static object, because of its dynamic nature. In particular, when someone queries an event stream for the "next" event, the provided answer will depend on the "query time".

In order to achieve our goal, we decided to model event streams using TCP connections [7, 8]. In particular, we employed a client-server architecture in which the stream source acts as a server and several miners, modeled as clients, can be connected to the same source. Figure 1 reports a graphical representation of such idea: the stream source forwards each event, represented as a box, to all the connected miners (clients) via TCP connections.

When a new event is emitted, a log fragment is encoded only containing that event and sent to the clients. Specifically, the log fragment contains only one trace which contains only the event emitted and is encoded in XML, using the XES standard [13]. It is important to note that, differently from what happens with static log files, events are not grouped anymore by process instance. Instead, each event is wrapped into different trace elements which have the same value for the "case id" attribute. The reconstruction of the complete trace is delegated to the clients.

The log fragment reported in Listing 1.1 represents an example of data written through the network connection.

[4] See `http://www.processmining.org` for more information.

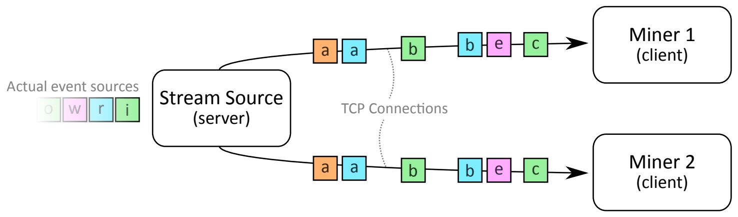


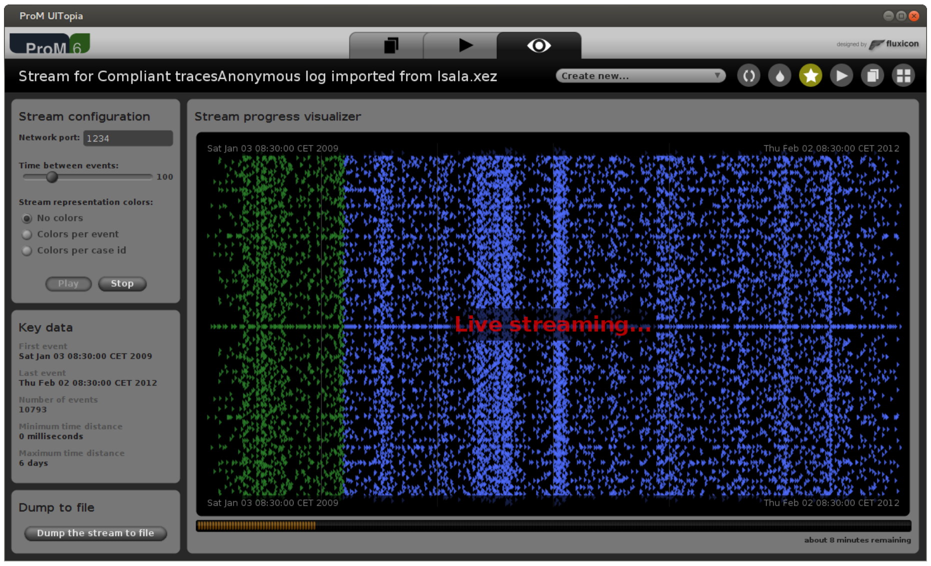
Fig. 1. Graphical representation of the stream architecture: the stream source is a *server*, and each miner is a *client* connected via TCP. Each square box represents an event. The background color of a box corresponds to the case id of the event.

```
<log openxes.version="1.0RC7" xes.features="nested-attributes"
    xes.version="1.0" xmlns="http://www.xes-standard.org/">
  <trace>
    <string key="concept:name" value="7235124248-6934584346" />
    <event>
      <string key="concept:name" value="Activity A" />
    </event>
  </trace>
</log>
```

Listing 1.1. Log fragment streamed over the network and encoded using XES.

We opted for this architecture since it is an extremely flexible solution and allows different computational nodes to communicate easily. Moreover, it is extremely easy, for an existing information system, to emit events to be analyzed using our implemented solution. In order to simulate specific stream scenarios, we use an *event streamer*, which takes as input a (finite) event log and simulates an event stream using the events contained in the log. Specifically, all the events of the log are sorted according to their timestamp. Then, each event is sent through the network. The time passing between two consecutive events is a parameter of the streamer. Clearly, the stream generated using this tool is not infinite, but it is still useful for testing purposes.

Fig. 2 reports a screenshot of the ProM implementation of the *event streamer*. On the right hand side of the screenshot there is a graphical representation of the stream: each colored point represents an event of the stream (green dots represent sent event whereas blue dots represent the events that still need to be sent). The x axis represents the time, the y axis has no specific semantic (simultaneous events are vertically distributed for readability purposes). The left hand side of the screenshot contains some configuration panels. In particular, it is possible to configure the network port that is used to accept incoming miners, the time passing between the emission of two consecutive events, and the control buttons to start and to stop emitting events on the network channel. Few other controllers are available as well. For example, it is possible to assign different colors to the events visualization according to the activity name or according to the case id.



Stream for Compliant tracesAnonymous log imported from Isala.xez

Network port: 1234

Time between events: 100

Stream representation colors:
- [✓ No colors]
- Colors per event
- Colors per case id

First event
Sat Jan 03 08:30:00 CET 2009

Last event
Thu Feb 02 08:30:00 CET 2012

Number of events
10793

Minimum time distance
0 milliseconds

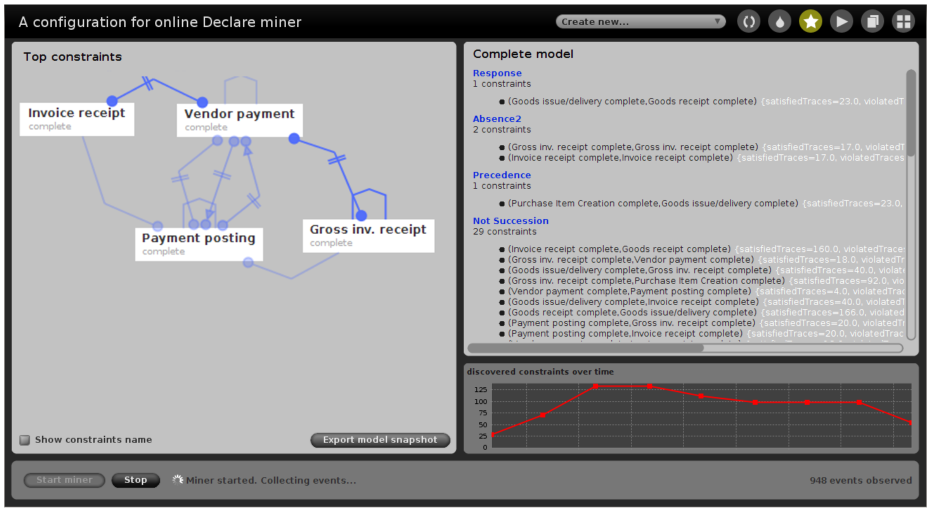
Maximum time distance
6 days

Fig. 2. Screenshot of the streamer interface.

3.2 The Declare Stream Miner

The ProM implementation of the Declare stream miner is fundamentally based on the "observer pattern": every time the stream receives a new event, different replayers update their internal status with the new observation. With a certain periodicity, either with respect to the number of events received or with respect to the time passed, the Declare model is updated, taking into account the new status recorded by each replayer.

Since the mined model is not static (it evolves over time), it is not possible to generate one model to be added to the ProM workspace. Therefore, we decided to implement a *dashboard* in the ProM toolkit. This dashboard (Fig. 3) shows the current status of the stream, together with some general historical information. The current behavior as detected from the event stream is shown in the form of a list of Declare rules that change over time. While the stream progresses, the visualization in the dashboard is periodically updated showing the currently valid Declare rules. The dashboard is composed of several parts: the bottom panel contains some general controllers, to turn on and off the miner, together with some general information about the number of received events. The main panel is vertically splitted: the left hand side part reports a graphical representation of the top 10 rules currently valid (the ones with the highest fulfillment ratio). In this panel, the value of the fulfillment ratio is encoded using different colors: brighter blue indicates the lowest values, darker blue indicates the highest values. The same panel contains two more controllers: one to export the currently displayed Declare model into the ProM workspace, and another one to attach the rule name to each connector in the model. The right hand side



Complete model

Response
1 constraints
- (Goods issue/delivery complete,Goods receipt complete) {satisfiedTraces=23.0, violatedT

Absence2
2 constraints
- (Gross inv. receipt complete,Gross inv. receipt complete) {satisfiedTraces=17.0, violatedT
- (Invoice receipt complete,Invoice receipt complete) {satisfiedTraces=17.0, violatedTraces

Precedence
1 constraints
- (Purchase Item Creation complete,Goods issue/delivery complete) {satisfiedTraces=23.0,

Not Succession
29 constraints
- (Invoice receipt complete,Goods receipt complete) {satisfiedTraces=160.0, violatedTrace
- (Gross inv. receipt complete,Vendor payment complete) {satisfiedTraces=18.0, violatedTr
- (Goods issue/delivery complete,Gross inv. receipt complete) {satisfiedTraces=40.0, violate
- (Gross inv. receipt complete,Purchase Item Creation complete) {satisfiedTraces=92.0, vio
- (Vendor payment complete,Payment posting complete) {satisfiedTraces=4.0, violatedTrac
- (Goods issue/delivery complete,Invoice receipt complete) {satisfiedTraces=40.0, violatedT
- (Goods receipt complete,Goods issue/delivery complete) {satisfiedTraces=166.0, violated
- (Payment posting complete,Gross inv. receipt complete) {satisfiedTraces=20.0, violatedTr
- (Payment posting complete,Invoice receipt complete) {satisfiedTraces=20.0, violatedTrac

Fig. 3. Screenshot of the stream miner dashboard.

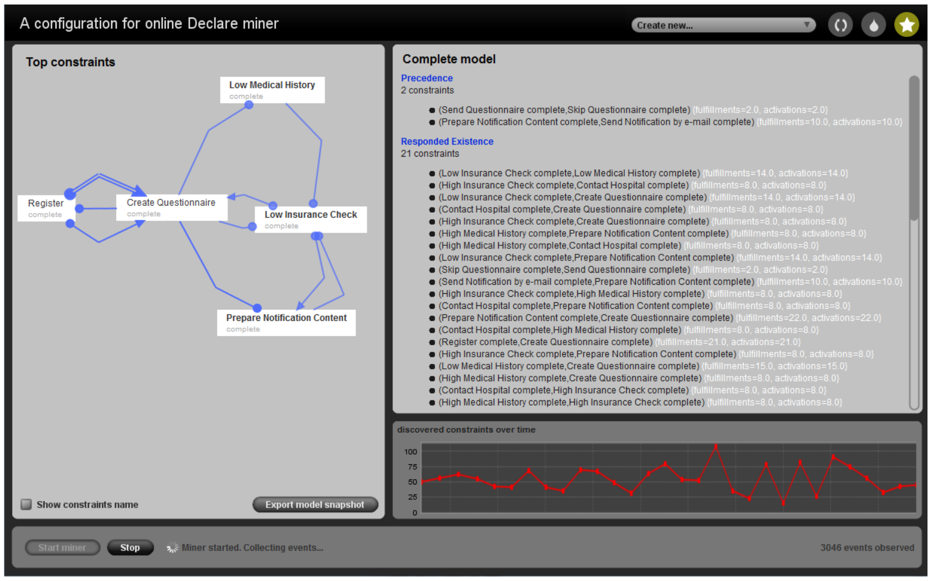
of the dashboard contains two panels: the one on the top contains a list with the complete set of discovered rules (with the corresponding statistics), the panel on the bottom part contains a line chart with some historical information. In our specific case, the chart shows the trend of the number of rules discovered over time, but several other statistics can easily be added as well (e.g., the average fulfillment ratio of the rules).

4 Case Study

For our experiment,[5] we have generated two synthetic logs ($\mathcal{L}_1$ and $\mathcal{L}_2$) by modeling two variants of the insurance claim process described in [5] in CPN Tools[6] and by simulating the models. The following characteristics are common to both process variants. Upon registration of a claim, a general questionnaire is sent to the claimant. In addition, a registered claim is classified as high or low. A cheque and acceptance decision letter is prepared in cases where a claim is accepted while a rejection decision letter is created for rejected claims. In both cases, a notification is sent to the claimant. Three modes of notification are supported, i.e., by email, by telephone (fax) and by postal mail. The case should be archived upon notifying the claimant. The case is closed upon completion of archiving task.

$\mathcal{L}_1$ contains 14,840 events and $\mathcal{L}_2$ contains 16,438 events. We merged the logs (four alternations of $\mathcal{L}_1$ and $\mathcal{L}_2$) using the *Stream Package*, publicly available

[5] The entire process movie generated in our experiment can be found at `http://youtu.be/9gbrhkSfRTc`.
[6] See `http://cpntools.org`.



A configuration for online Declare miner

Top constraints

Complete model

Precedence
2 constraints

- (Send Questionnaire complete,Skip Questionnaire complete) {fulfillments=2.0, activations=2.0}
- (Prepare Notification Content complete,Send Notification by e-mail complete) {fulfillments=10.0, activations=10.0}

Responded Existence
21 constraints

- (Low Insurance Check complete,Low Medical History complete) {fulfillments=14.0, activations=14.0}
- (High Insurance Check complete,Contact Hospital complete) {fulfillments=8.0, activations=8.0}
- (Low Insurance Check complete,Create Questionnaire complete) {fulfillments=14.0, activations=14.0}
- (Contact Hospital complete,Create Questionnaire complete) {fulfillments=8.0, activations=8.0}
- (High Insurance Check complete,Create Questionnaire complete) {fulfillments=8.0, activations=8.0}
- (High Medical History complete,Prepare Notification Content complete) {fulfillments=8.0, activations=8.0}
- (High Medical History complete,Contact Hospital complete) {fulfillments=8.0, activations=8.0}
- (Low Insurance Check complete,Prepare Notification Content complete) {fulfillments=14.0, activations=14.0}
- (Skip Questionnaire complete,Send Questionnaire complete) {fulfillments=2.0, activations=2.0}
- (Send Notification by e-mail complete,Prepare Notification Content complete) {fulfillments=10.0, activations=10.0}
- (High Insurance Check complete,High Medical History complete) {fulfillments=8.0, activations=8.0}
- (Contact Hospital complete,Prepare Notification Content complete) {fulfillments=8.0, activations=8.0}
- (Prepare Notification Content complete,Create Questionnaire complete) {fulfillments=22.0, activations=22.0}
- (Contact Hospital complete,High Medical History complete) {fulfillments=8.0, activations=8.0}
- (Register complete,Create Questionnaire complete) {fulfillments=21.0, activations=21.0}
- (High Insurance Check complete,Prepare Notification Content complete) {fulfillments=8.0, activations=8.0}
- (Low Medical History complete,Create Questionnaire complete) {fulfillments=15.0, activations=15.0}
- (High Medical History complete,Create Questionnaire complete) {fulfillments=8.0, activations=8.0}
- (Contact Hospital complete,High Insurance Check complete) {fulfillments=8.0, activations=8.0}
- (High Medical History complete,High Insurance Check complete) {fulfillments=8.0, activations=8.0}

Miner started. Collecting events...

3046 events observed

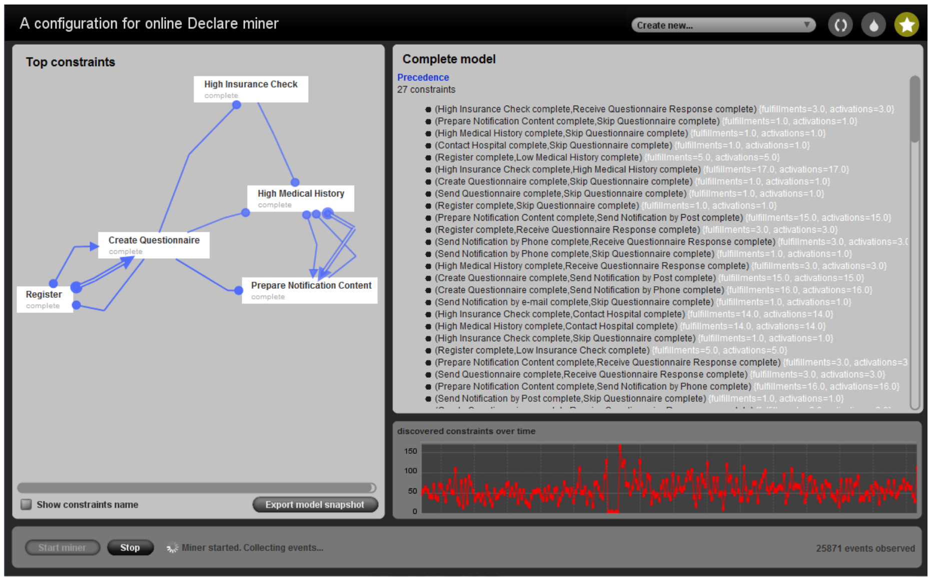
Fig. 4. Screenshot of the visualizer (frame from $\mathcal{L}_1$).

in the ProM repositories. The same package has been used to transform the resulting log into an event stream. The event stream contains 250,224 events and has three concept drifts (one for every switch from $\mathcal{L}_1$ to $\mathcal{L}_2$).

The event stream has been simulated with the event streamer. The events, received by the Declare stream miner has been processed to discover Declare rules that hold with a fulfillment ratio of 1. A Declare model is generated every 100 milliseconds, i.e., every 100 milliseconds the model shown through the visualizer is refreshed and the process movie progresses with a new frame.

In Fig. 4, a frame of the process movie is shown generated by simulating the event stream. The figure shows one of the frames of the movie when the events from $\mathcal{L}_1$ are sent to the miner. The frame indicates that at that point in time during the stream evolution the Declare rules involving claims classified as "low" are prevalent in terms of fulfilment ratio. As already mentioned, this view can be exported to create a snapshot of the process behavior at that point in time during the stream progression. Fig. 5 shows one of the frames of the movie when the events from $\mathcal{L}_2$ are sent to the miner. In this case, the Declare rules involving claims classified as "high" correspond to a higher fulfilment ratio.

In Fig. 6, the entire trend of the number of discovered constraints from the beginning to the end of the process movie is shown. The concept drifts when passing from one of the two logs to the other are evident and indicated as peaks in the curve.



Complete model

Precedence

27 constraints

- (High Insurance Check complete,Receive Questionnaire Response complete) {fulfillments=3.0, activations=3.0}
- (Prepare Notification Content complete,Skip Questionnaire complete) {fulfillments=1.0, activations=1.0}
- (High Medical History complete,Skip Questionnaire complete) {fulfillments=1.0, activations=1.0}
- (Contact Hospital complete,Skip Questionnaire complete) {fulfillments=1.0, activations=1.0}
- (Register complete,Low Medical History complete) {fulfillments=5.0, activations=5.0}
- (High Insurance Check complete,High Medical History complete) {fulfillments=17.0, activations=17.0}
- (Create Questionnaire complete,Skip Questionnaire complete) {fulfillments=1.0, activations=1.0}
- (Send Questionnaire complete,Skip Questionnaire complete) {fulfillments=1.0, activations=1.0}
- (Register complete,Skip Questionnaire complete) {fulfillments=1.0, activations=1.0}
- (Prepare Notification Content complete,Send Notification by Post complete) {fulfillments=15.0, activations=15.0}
- (Register complete,Receive Questionnaire Response complete) {fulfillments=3.0, activations=3.0}
- (Send Notification by Phone complete,Receive Questionnaire Response complete) {fulfillments=3.0, activations=3.
- (Send Notification by Phone complete,Skip Questionnaire complete) {fulfillments=1.0, activations=1.0}
- (High Medical History complete,Receive Questionnaire Response complete) {fulfillments=3.0, activations=3.0}
- (Create Questionnaire complete,Send Notification by Post complete) {fulfillments=15.0, activations=15.0}
- (Create Questionnaire complete,Send Notification by Phone complete) {fulfillments=16.0, activations=16.0}
- (Send Notification by e-mail complete,Skip Questionnaire complete) {fulfillments=1.0, activations=1.0}
- (High Insurance Check complete,Contact Hospital complete) {fulfillments=14.0, activations=14.0}
- (High Medical History complete,Contact Hospital complete) {fulfillments=14.0, activations=14.0}
- (High Insurance Check complete,Skip Questionnaire complete) {fulfillments=1.0, activations=1.0}
- (Register complete,Low Insurance Check complete) {fulfillments=5.0, activations=5.0}
- (Prepare Notification Content complete,Receive Questionnaire Response complete) {fulfillments=3.0, activations=3
- (Send Questionnaire complete,Receive Questionnaire Response complete) {fulfillments=3.0, activations=3.0}
- (Prepare Notification Content complete,Send Notification by Phone complete) {fulfillments=16.0, activations=16.0}
- (Send Notification by Post complete,Skip Questionnaire complete) {fulfillments=1.0, activations=1.0}

Fig. 5. Screenshot of the visualizer (frame from $\mathcal{L}_2$).

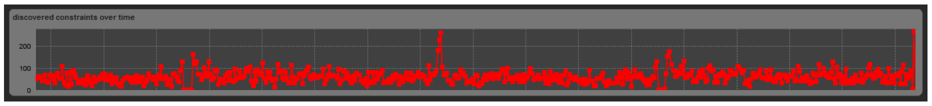


Fig. 6. Trend of the number of discovered constraints from the beginning to the end of the process movie.

5 Conclusion

This paper proposes a tool for the dynamic visualization of process models discovered through a technique for the online discovery of Declare rules described in our previous work. The different models are discovered in different evaluation points from an event stream. These models represent different frames of a process movie. A process movie is a way of dynamically visualizing the results of an online process discovery technique like the ones implemented in the Declare stream miner. Our experimentation shows that the tool is able to reproduce graphically the evolution of process cases encoded in an event stream. In addition, concept drifts are also graphically captured through the trend line representing the number of discovered Declare rules over time.

Acknowledgment The work by Andrea Burattin is supported by the Eurostars-Eureka project PROMPT (E!6696).

References

1. van der Aalst, W.: Process mining: Overview and opportunities. ACM Trans. Manage. Inf. Syst. 3(2), 7:1–7:17 (Jul 2012)
2. van der Aalst, W., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. Computer Science - R&D pp. 99–113 (2009)
3. Aggarwal, C.: Data Streams: Models and Algorithms, Advances in Database Systems, vol. 31. Springer US, Boston, MA (2007)
4. Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B.: MOA: Massive Online Analysis Learning Examples. Journal of Machine Learning Research 11, 1601–1604 (2010)
5. Bose, R.J.C.: Process Mining in the Large: Preprocessing, Discovery, and Diagnostics. Ph.D. thesis, Eindhoven University of Technology (2012)
6. Burattin, A., Maggi, F., van der Aalst, W., Sperduti, A.: Techniques for a Posteriori Analysis of Declarative Processes. In: EDOC. pp. 41–50 (2012)
7. Burattin, A., Sperduti, A., Aalst, W.v.d.: Heuristics Miners for Streaming Event Data. CoRR abs/1212.6383 (2012)
8. Burattin, A., Sperduti, A., Aalst, W.v.d.: Control-flow Discovery from Event Streams. In: Proceedings of the IEEE Congress on Evolutionary Computation. IEEE, (to appear) (2014)
9. Da San Martino, G., Navarin, N., Sperduti, A.: A Lossy Counting Based Approach for Learning on Streams of Graphs on a Budget. In: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence. pp. 1294–1301. AAAI Press (2012)
10. Gaber, M.M., Zaslavsky, A., Krishnaswamy, S.: Mining Data Streams: a Review. ACM Sigmod Record 34(2), 18–26 (Jun 2005)
11. Gama, J.a.: Knowledge Discovery from Data Streams, Chapman & Hall/CRC Data Mining and Knowledge Discovery Series, vol. 20103856. Chapman and Hall/CRC (May 2010)
12. Golab, L., Özsu, M.T.: Issues in Data Stream Management. ACM SIGMOD Record 32(2), 5–14 (Jun 2003)
13. Günther, C., Verbeek, H.: XES Standard Definition. www.xes-standard.org (2009), `http://www.xes-standard.org/`
14. Kupferman, O., Vardi, M.: Vacuity Detection in Temporal Model Checking. Int. Journal on Software Tools for Technology Transfer pp. 224–233 (2003)
15. Maggi, F.M., Burattin, A., Cimitile, M., Sperduti, A.: Online process discovery to detect concept drifts in ltl-based declarative process models. In: On the Move to Meaningful Internet Systems: OTM 2013 Conferences. Proceedings. pp. 94–111 (2013)
16. Manku, G.S., Motwani, R.: Approximate Frequency Counts over Data Streams. In: VLDB. pp. 346–357 (2002)
17. Montali, M., Pesic, M., van der Aalst, W.M.P., Chesani, F., Mello, P., Storari, S.: Declarative Specification and Verification of Service Choreographies. ACM Transactions on the Web 4(1) (2010)
18. Pesic, M.: Constraint-Based Workflow Management Systems: Shifting Controls to Users. Ph.D. thesis, Beta Research School for Operations Management and Logistics, Eindhoven (2008)
19. Verbeek, E., Buijs, J., van Dongen, B., van der Aalst, W.: Prom 6: The process mining toolkit. In: Demo at the 8th International Conference on Business Process Management (BPM 2010) (2010)
20. Widmer, G., Kubat, M.: Learning in the Presence of Concept Drift and Hidden Contexts. Machine Learning 23(1), 69–101 (1996)