MQTT-XES: Real-time telemetry for process event data

Burattin, Andrea; Eigenmann, Martin; Seiger, Ronny; Weber, Barbara

Published in:
CEUR Workshop Proceedings

Publication date:
2020

Document Version
Publisher's PDF, also known as Version of record

MQTT-XES: Real-time Telemetry for Process Event Data

Andrea Burattin[1], Martin Eigenmann[2], Ronny Seiger[2], and Barbara Weber[2]

[1] Software and Process Engineering, Technical University of Denmark, 2800 Kgs. Lyngby, Denmark
[2] Institue of Computer Science, University of St. Gallen, 9000 St. Gallen, Switzerland

Abstract. This demo paper presents an infrastructure to enable real-time monitoring of process events (i.e., telemetry). The infrastructure relies on the MQTT protocol which ensures minimum logging overhead. The paper presents a Java library for producing (i.e., logging) and consuming events, built on top of HiveMQ. Additionally, a prototype dashboard to display basic statistics is reported and described.

1 Introduction

Recording happenings of interest in the context of process executions is becoming more and more important. Specifically, the spread of process mining techniques [2] has paved the way to an unprecedented set of opportunities ranging from discovering the processes actually being executed (as opposed to the intended ones), calculating the conformity of executions, identifying bottlenecks, predicting the outcomes of processes, suggesting resource allocations, etc.

In its typical form, process mining consumes offline recordings of historical executions (the so called "event logs"). With this paper we suggest a new family of tools for logging events in real-time (i.e., online) and share them with "interested" entities. Such type of remote logging, also referred to as "*telemetry*", represents a key component for the deployment of online process mining techniques [5] that, in turn, enables to extract knowledge that can be immediately exploited with almost no delay. For example, let's consider a conformance checking problem in a hospital: the process managers would like to know immediately whether patients are treated according to standard protocols (i.e., the reference processes) or not. With offline techniques, the delay between when the violation happened and when the managers are notified depends on how often logs are extrapolated and processed (this time span can be in the range of days up to months or years). With real-time process mining systems, violations would be reported immediately after they occurred, thus giving the process manager time to compensate the issue. In the rest of the paper we present some general ideas about the principles behind the developed artifacts (cf. Sec. 2) as well as some technical details (cf. Sec. 3).

2 Innovation and Related Work

To achieve the goal mentioned in the previous section, i.e., a real-time process mining system, it is necessary to build a stream processing architecture suit-

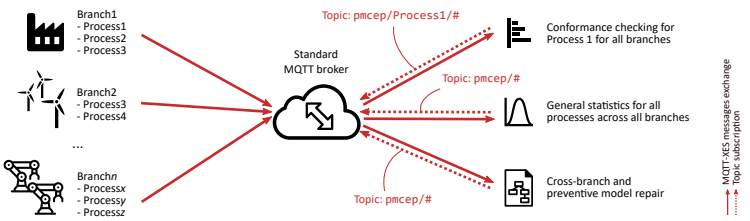

Fig. 1. Graphical representation of possible use-cases of the contribution of the paper.

able for distributed settings and capable of high throughput. With this paper we contribute towards this goal by suggesting a software architecture for *unified logging* (i.e., an append only, ordered, distributed log) [6]. Specifically, we propose to utilize the Message Queuing Telemetry Transport (MQTT) protocol [7, 3] for tracking the execution of business process activities in real-time. MQTT has been specifically devised to operate using a publish/subscribe protocol and, in particular, the key idea was to support the logging and the telemetry for the Internet of Things, i.e., devices where the computing power is low. Consistently with such rationale, most of the complexity is delegated to the "broker": the remote component in charge of receiving the events from the producers and forwarding them to the interested consumers. Though other protocols are possible (e.g., XMPP, JMS, AMQP), MQTT represents the optimal choice to handle the real-time logging of activities being executed, since the overhead is reduced to the minimum. MQTT is a topic-based publish/subscribe protocol and in this paper we propose a specific structure for the organization of the topics. By exploiting such a structure, the logging overhead is reduced even more since, even with no payload, the mere fact that an event is published on a certain topic is informative for most of the common process mining tasks. This aspect makes MQTT-XES truly interoperable (relying on the payload makes the protocol not really interoperable, since the payload is just a byte array [7, 3]).

In Fig. 1, we sketched an idea concerning possible usages of MQTT-XES. For example, we might have several branches of the organization, each of them running different processes, all generating events that are sent to the same MQTT broker. Before reaching the broker it could be that some of these events are actually pre-processed, for example to reach the same abstraction level. From the broker, events can be forwarded to several subscribed consumers, each of them in charge of different aspects. These consumers will rely on two important aspects of the data streams: *(i)* all events are available in real time, and *(ii)* events are referring not just to individual processes but to the whole organization. Therefore, it will be possible to construct dashboards monitoring the whole system (as in "all the processes running across the whole organization"), calculating conformity for the same process being executed at several branches, or defining model repair algorithms capable of improving processes executed at a branch based on observations coming from a different branch. When logs are stored locally in an offline fashion, all these use cases are not feasible. The contribution of this paper is three-fold. First, we propose a specific schema for the topic of published

events that reflects the typical structure of the XES standard [1]. After that we present a Java library to log the executions of events and, finally, we present a dashboard to consume events and report some statistics.

The idea of streaming events, in the BPM context, is not new in the literature [5] however, so far, no unified system has been devised for this purpose. An approach based on a publish/subscribe protocol is available [8] but is is accessible just within a single instance of the ProM toolkit, making it not suitable for distributed settings. An alternative approach [4], exclusively based on TCP connections, leveraged the idea of transmitting small fragments of XES logs (i.e., each event is an `XLog` with one `XTrace` with one `XEvent`). However, the lack of a unified protocol and the verbosity of the log fragments make this TCP-based system not suitable to reach high throughput and interoperability.

3 Technical Details and Maturity

In MQTT, topics represent "addresses" where events can be sent to. They are typically used to define scopes and, therefore, are used by consumers to receive only relevant events. Topics are structured in "hierarchies" which are defined using levels and level separators. An example of a topic where events referring to a room temperature are published is the following: `lyngby/B322/R212/temp`. We can expect that each event in such topic will contain as payload, a representation of the temperature for room `R212` of building `B322` of the `lyngby` campus. Consumers, therefore, can decide to subscribe to specific topics, for example to receive all temperature readings for that room. Consumers can also decide to receive events from whole hierarchies of topics by using wildcards. Two wildcards are commonly used: a single-level (indicated with `+`) and multi-level wildcard (`#`). With the single level wildcard a consumer could subscribe to all temperature events of building `B322` (so temperatures of all offices): `lyngby/B322/+/temp`. With the multi-level wildcard it is possible to subscribe to all events happening at the `lyngby` campus: `lyngby/#`.

In this paper we suggest a specific hierarchy of the topics that will allow tracking process event data. The structure is the following:

`BASE/[SOURCE_ID]/[PROCESS_INSTANCE_ID]/[ACTIVITY_NAME]`

In this structure, `BASE` refers to a predefined name useful to identify all events part of the activity monitoring initiative. `[SOURCE_ID]` refers to the name of the source of the event. According to the XES standard this would be the identifier of the log. Practically speaking, this could refer to the name of the process and the branch where the process is performed. `[PROCESS_INSTANCE_ID]` is the identifier of the case id and `[ACTIVITY_NAME]` is the name of the activity executed. With such a structure, topics are "reserved" for each process instance and for each activity name.[3] Therefore, the mere fact that an event was published in a topic indicates that the corresponding activity happened, with no need to check the actual payload of the event. This makes the whole system extremely lightweight

[3] Note that in MQTT topics do not need to be initialized however, conceptually, the namespace of topics would reflect the events universe [2].

Listing 1.1. Example of how to use the Java API to send an MQTT-XES event.

```
1 // Specification of the client with broker and root topic level
2 XesMqttSerializer client = new XesMqttSerializer("broker.hivemq.com", "BASE");
3 // Construct of the MQTT event
4 XesMqttEvent event =
5     new XesMqttEvent("source-id", "case-id", "activity")
6     .addTraceAttribute("name", "value").addEventAttribute("name", "value")
7     .removeTraceAttribute("dummy"); // Manipulation using method chaining
8 // Connection to the broker and event sending
9 client.connect(); client.send(event); client.disconnect();
```

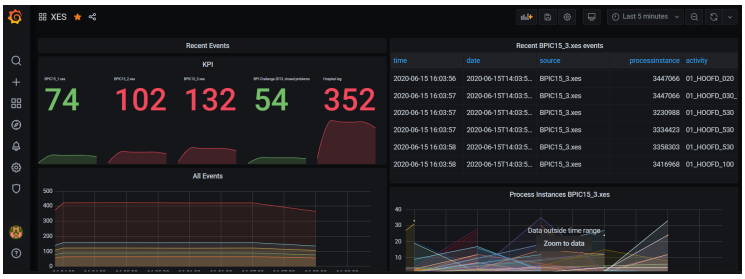

Fig. 2. The developed dashboard showing some basic statistics for all events

and truly interoperable (the topic name must be an UTF-8 Encoded string [3]). With the current implementation of MQTT-XES, it is also possible to define attributes at the process instance and at the event levels, which are encoded into the payload as UTF-8 string containing a JSON object.

To simplify the usage of MQTT-XES, a Java library has been developed (based on HiveMQ, cf. `https://www.hivemq.com/`). This library, which is aware of the typical BPM concepts, can be used to generate and publish events as well as to consume them. An example of usage of the library to send events is reported in Listing 1.1: in line 2 the basic configuration is provided, namely the broker's host address (in this case, we used the the public broker of HiveMQ[4]) and the `BASE` of the topic. Then, in lines 4-7, the actual event to be sent is defined. The constructor (line 5) requires an identifier for the source (e.g., a concatenation of the name of the branch and the name of the process), the process instance identifier and the name of the activity. The following lines (6-7) show possible ways of adding attributes for the process instance or for the event. In line 9, finally, the event is sent to the broker.

To subscribe to the events it is possible to use all standard MQTT-consumers. The Java library we developed can also be used to consume events using a callback mechanism[5]. For this work we also developed a dashboard, specifically targeting events generated for MQTT-XES. The dashboard is built using Grafana (see `https://grafana.com/`) for visualisation and PostgreSQL as storage back-

[4] In their website they state that *"Testing and usage is for free but please do not use it for sensitive information because everybody is allowed to subscribe to every topic"*.

[5] For a tutorial on how to use the library see `https://github.com/pmcep/mqtt-xes`.

end. The MQTT event stream is collected and subsequently persisted to the datastore. Grafana allows the visualisation of this data based on user defined queries. These queries can also include filter and aggregation operations and thus, the dashboard is capable of displaying a wide variety of metrics of multiple streams in a unified manner. Although the current state of dashboard displays several metrics, it is still an early prototype showing a proof of concept.

4 Conclusions

The work presented in this paper can be summarized as an infrastructure to enable real-time monitoring of process events. This infrastructure relies on the MQTT protocol, which is extremely lightweight and therefore imposes minimum overhead. Exploiting the proposed structure of the MQTT topics, real interoperability is achieved (since no payload is formally necessary). The provided Java library can be used to simplify the generation of the messages as well as their consumption. The dashboard accompanying the paper can also be used to immediately consume the events and get an idea of how the system is behaving. Enabling BPM-engines to generate events using MQTT-XES comes with several advantages and the ability to overcome some major limitations of existing BPM technologies. Specifically, the ability to extend the mining beyond the single process to focus on entire systems or investigate a single process across different branches of an organization will pave the way to a whole new family of techniques and algorithms, yet to be developed. All of that in an online fashion, which enables rapid reaction times.

A screencast showing the tools is available at `https://youtu.be/-jrkd6kl8Nw`. A brief tutorial on how to use the APIs is available on the GitHub page of the tools at `https://github.com/pmcep/mqtt-xes/` and `https://github.com/pmcep/mqtt-xes-dashboard`, where it is also possible to download all the source code. An installation of the dashboard is available at `http://mqttxes.ics.unisg.ch/d/PkSsD3mGk/xes`. MQTT-XES is used to stream the events of some BPI Challenges (i.e., 2011, 2013-closed, 2015-m1, 2015-m2, and 2015-m3) to the broker `broker.hivemq.com:1883`.

References

1. IEEE Standard for eXtensible Event Stream (XES) for Achieving Interoperability in Event Logs and Event Streams. IEEE Std 1849-2016 pp. 1–50 (2016)
2. van der Aalst, W.: Process Mining: Data Science in Action. Springer (2016)
3. Banks, A., Briggs, E., Borgendale, K., Gupta, R.: MQTT V. 5.0. OASIS Std. (2019)
4. Burattin, A.: Process Mining Techniques in Business Environments, LNBIP, vol. 207. Springer (2015)
5. Burattin, A.: Streaming Process Discovery and Conformance Checking. In: Encyclopedia of Big Data Technologies (2019)
6. Dean, A., Crettaz, V.: Event Streams in Action. Manning (2019)
7. Hunkeler, U., Truong, H.L., Stanford-Clark, A.: MQTT-S - A publish/subscribe protocol for Wireless Sensor Networks. In: Proc. of COMSWARE. pp. 791–798 (2008)
8. van Zelst, S.J., Burattin, A., van Dongen, B.F., Verbeek, H.M.W.: Data Streams in ProM 6: A Single-node Architecture. In: BPM Demo Sessions 2014 (2014)