Data Streams in ProM 6: A Single-node Architecture

S.J. van Zelst[1], A. Burattin[2], B.F. van Dongen[1] and H.M.W. Verbeek[1]

[1] Eindhoven University of Technology
{s.j.v.zelst,b.f.v.dongen,h.m.w.verbeek}@tue.nl
[2] University of Padova
burattin@math.unipd.it

Abstract. Process mining is an active field of research that primarily builds upon data mining and process model-driven analysis. Within the field, static data is typically used. The usage of dynamic and/or volatile data (i.e. real-time streaming data) is very limited. Current process mining techniques are in general not able to cope with challenges posed by real-time data. Hence new approaches that enable us to apply process mining on such data are an interesting new field of study. The ProM-framework that supports a variety of researchers and domain experts in the field has therefore been extended with support for data-streams. This paper gives an overview of the newly created extension that lays a foundation for integrating streaming environments with ProM. Additionally a case study is presented in which a real-life online data stream has been incorporated in a basic ProM-based analysis.

1 Introduction

We assume the reader to be acquainted with the field of process mining and refer to [1] for an elaborate overview of the field. Henceforth we turn our focus towards the ProM framework[3] [2, 5] which since its introduction greatly enhanced the availability of several process mining techniques. Current analyses within ProM are primarily based on *static* data in the form of "transaction logs" originating from one or many information systems. The current state of art in computer science technology enables us to both store and generate large quantities of data at tremendous throughput rates. Within this context there has only been a limited amount of research conducted towards the integration of streaming data and process mining.

Present approaches able to deal with streams in ProM [3, 4, 7] are based on Java sockets. In these implementations a stream is represented by a network connection using TCP. The stream's source is a network-server that accepts connections from stream-miners. After a network connection has been established between a miner and a stream-source, the latter starts "emission" of events into the channel.

The available implementations do not provide any means of standardization of event-like data-packets send over a stream apart from sending multiple single events containing log-fragments using XES [6]. The use of TCP limits the maximum throughput speed of the stream by the computational speed of the slowest connected analysis

[3] http://www.promtools.org/

Fig. 1: A generalized overview of the ProM-stream architecture.

tool, resulting in a very strong coupling between sending and receiving parties. In practical applications, events are emitted independent of the analysis performed on them and therefore it is desirable to decouple these entities as much as possible, even in a single node setting. An additional problem with the aforementioned implementations is limited reuse-ability of code.

In this demo paper we present a standardized approach to the use of streaming data within ProM. In that sense the framework has been extended with basic support for handling streaming data. The newly created extension simplifies application of process mining on dynamic and volatile sources of data. The general aim of this paper is to create awareness for the adoption of handling streaming data within process mining.

2 Architecture

2.1 Entities

The basic architecture chosen within ProM w.r.t. streaming is mainly intended to help developers in structuring stream handling[4]. The Publisher-Subscribe pattern acts as a fundamental basis of using streams in ProM. A basic UML-diagram of the most important architectural components is shown in Fig. 1.

Streams, **publishers** and **subscribers** are "ProM Objects". This means that they are visible and selectable objects in the ProM *workspace*. A publisher consists of a set of **authors**. Each author writes data onto a dedicated stream. A subscriber consists of a set of **readers**. Each reader reads data from a dedicated stream. Different readers are able to connect to the same stream. The stream entity will in fact create a dedicated channel for each connected reader. In this way we achieve maximal decoupling between sending and receiving party.

The publisher and subscriber entities are completely standardized and independent of the data-type sent over a stream. On the other hand, authors, readers and streams are strongly typed. Some of these entities are standardized and/or predefined (e.g. the notion of an `Event Stream`, a standardized `XLog` to `Event Stream` author etc.).

2.2 Data stream handling

Given an external data stream/source of some arbitrary type. An author's task is to translate this stream/source into a ProM stream object. An author is part of a publisher instance which itself is part of a ProM-plugin mapping the source data (stream) to an

[4] Handling a "stream" on a single node just concerns the (re)allocation of in-memory objects to several threads/workers.

$(n+1)$-tuple consisting of the publisher instance and n ProM stream objects, given that the publisher consists of n authors.

A preliminary task w.r.t. ProM stream creation is determining of what type the ProM stream should be. If a non-existing type is preferred, it is possible to create a new typed data-stream.

Once the external data stream/source is translated to a ProM stream object, it is ready for analysis. Readers are in general designated with this task. The analysis can be of any type ranging from just counting the total number of data-packets sent to mining an organizational-network etc. A reader is part of a subscriber which itself is part of a ProM plug-in returning just the subscriber instance.

A reader may have one or many result value(s) (e.g. a reconstructed log, a series of mined process models etc.). These results are however not return values of the reader's designated ProM plugin. They can be either visualized (section 3) or added to the ProM workspace in a later phase.

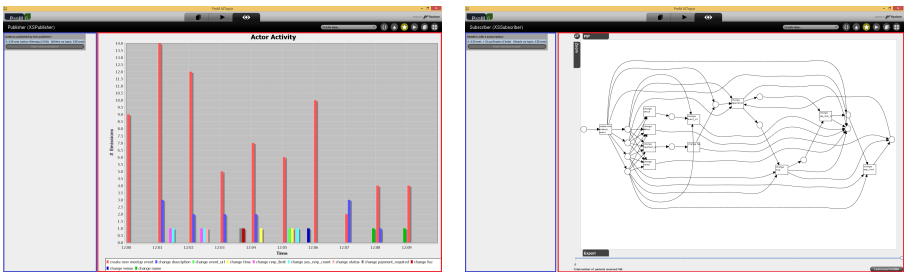
3 User Interface

As indicated, streams, subscribers and publishers act as ProM objects. Both publishers and subscribers have partly standardized graphical user interfaces.

The standardized publisher visualization is depicted in figure 2a. The visualization contains two separate panes. The left pane of the visualization (blue outlined in figure 2a) is a standardized visualization, currently only consisting of a list of active authors within the publisher. Underneath the list a button is located that allows the user to start the currently selected author.

The right-hand pane is customizable and author specific. Whenever a publisher contains more than one authors, selecting a different author from the list will result in a different visualization in the right-hand pane.

The standardized subscriber visualization is very similar w.r.t. the publisher visualization. Again the visualization contains two separate panes, one standardized pane depicting all the readers within the subscriber entity and one custom pane depicting the reader's visualization. An example screen-shot of a subscriber visualization is depicted in figure 2b.



(a) ProM publisher visualization (b) ProM subscriber visualization

Fig. 2: ProM screen-shots of both a publisher and subscriber visualization

4 Case Study

As an explanatory case we have used the ProM Stream infrastructure on real and live event data. The data used as an input was the open event stream of `http://www.meetup.com`[5]. A more elaborate demonstration of the case study is published on the first author's scientific web page[6].

`meetup.com` is a website where people can plan *happenings*. Within `meetup.com` happenings are called *events*. However, to avoid ambiguity w.r.t. the well-defined notion of events within process mining we will use the term *happenings* when we refer to meetup events. An example of a happening is a jam-session for musicians or an afternoon walk through Central Park in New York. A happening has several parameters (`name`, `status`, `venue` etc.). A user can create a happening and later-on change some parameters of the happening (for example, after creating a happening we decide to increase the total number of accepted guests). Every data-manipulation of a happening which has a public access level will be published on the `meetup.com` event stream.

The goal of the performed case study is to repeatedly run a process discovery algorithm over the event stream. In this way we can identify whether there is any behavioral pattern w.r.t. creation and manipulation of happenings. As the case study is purely of a demonstrative fashion we have not incorporated any frequent event set mining or memory reducing approach. We just collect "happening data packets" from the external stream, classify them within traces and store them. On a fixed regular basis (and if the user specifies to do so) the collected data will be converted to a log and passed to the α-miner. Note that it is very unlikely that there is actually a behavioral pattern involved in changing happenings.

For execution of the case study, two plug-ins are constructed. The "`XSEventStream Publisher (meetup.com/JSON)`" plug-in creates a publisher that consists of one author that takes `meetup.com`'s `JSON`-stream as an input and converts it to a standardized event stream (called `XSEvent`). The "`XSEventStream Subscriber (XSEvent -> PetriNet | α-Miner)`" plug-in creates a subscriber that consists of one reader that takes an event-stream (`XSEvent`) as an input. After a series of messages received, the reader's visualization will be updated with the latest process representation.

5 Future Work

Currently the architecture only consists of input and output nodes w.r.t. streams (e.g. authors and readers contained in publishers and subscribers respectively). It is however likely that there might be a need for entities that comprise both functionalities. It might be the case that we are only interested in emitting just a fraction of all events on the stream (i.e. filtering). Therefore within the future a new element might be introduced in the form of a "hub" which can consist of both a set of readers and a set of authors.

The architecture presented concerns a single-node implementation. If data throughput speed is extremely high using a single node might not be the most effective solution

[5] `http://www.meetup.com/meetup_api/docs/2/open_events/`

[6] `http://www.win.tue.nl/~svzelst/publications/2014_bpm_haifa_streams_in_prom/screencast/`

(apart from the necessity of applying frequent item-item/sequence mining). In the future, the architecture and implementation should be extended to support a multi-node setting as well. In this way we can assign separate nodes to perform stream production/-conversion, stream filtering, stream reading etc.

Although we have standardized several objects within ProM from an architectural perspective we have not yet spend a lot of attention in standardizing event streams. A future goal is to incorporate the notion of "streamed events" within the OpenXES standard such that it can be adopted even outside the context of ProM.

6 Conclusion

The newly presented single-node stream extension of ProM enables researchers and business users in the field to also apply process mining on streaming-data instead of solely relying on static data. The extension poses several new standardized architectural concepts and associated implementations aimed at re-usability and simplicity with respect to stream-handling within ProM. The case study shows that we are now able to effectively use (real) data-streams in the ProM framework.

Using the lessons learned in development of the current implementation, new challenges such as a multi-node setting and the notion of hubs can be tackled which will hopefully lead to a full integration of streaming-data analysis within the field of process mining.

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

References

1. W.M.P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer Publishing Company, Incorporated, 1st edition, 2011.
2. W.M.P. van der. Aalst, B.F. van Dongen, C.W. Günther, R.S. Mans, A.K. Alves de Medeiros, A. Rozinat, V. Rubin, M.S. Song, H.M.W. Verbeek, and A.J.M.M. Weijters. Prom 4.0: Comprehensive support for real process analysis. In J. Kleijn and A. Yakovlev, editors, *Petri Nets and Other Models of Concurrency – ICATPN 2007*, volume 4546 of *Lecture Notes in Computer Science*, pages 484–494. Springer Berlin Heidelberg, 2007.
3. A. Burattin, A. Sperduti, and W.M.P. van der Aalst. Heuristics Miners for Streaming Event Data. *CoRR*, abs/1212.6383, 2012.
4. A. Burattin, A. Sperduti, and W.M.P. van der Aalst. Control-flow Discovery from Event Streams. In *Proceedings of the IEEE Congress on Evolutionary Computation*. IEEE, (to appear), 2014.
5. B.F. van Dongen, A.K. Alves de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The prom framework: A new era in process mining tool support. In G. Ciardo and P. Darondeau, editors, *Applications and Theory of Petri Nets 2005*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer Berlin Heidelberg, 2005.
6. C.W. Günther and H.M.W. Verbeek. XES Standard Definition. www.xes-standard.org, 2009.
7. F.M. Maggi, A. Burattin, M. Cimitile, and A. Sperduti. Online Process Discovery to Detect Concept Drifts in LTL-Based Declarative Process Models. In *Proceedings of the OTM 2013 Conferences*, pages 94–111. Springer Berlin Heidelberg, 2013.