Integrated, Ubiquitous and Collaborative Process Mining with Chat Bots

Andrea Burattin[0000−0002−0837−0183]

Software and Process Engineering
Technical University of Denmark
andbur@dtu.dk

Abstract. Within the process mining field we are witnessing a tremendous growth of applications and development frameworks available to perform data analyses. Such growth, which is very positive and desirable, comes with the cost of learning each new tool and difficulties in integrating different systems in order to complement the analyses. In addition, we are noticing the lack of tools enabling collaboration among the users involved in a project. Finally, we think it would be highly recommended to enable ubiquitous processing of data. This paper proposes a solution to all these issues by presenting a chat bot which can be included in discussions to enable the execution of process mining directly from the chat.

Keywords: Process Mining · Chat bots · Ubiquitous computing.

1 Introduction

Process mining [1] has been gaining more attention over the previous years, becoming an impactful discipline, both in terms of research maturity and applications/implementations availability. It comprises the analysis of data referring to business process events and, within itself, it is possible to identify several sub-disciplines, such as control-flow discovery, conformance checking and process extension [8].

With a multitude of tools available, both as standalone, web applications and libraries, process mining projects can now be completed taking advantage of a broad range of possibilities. In particular, just by focusing on the open source tools, one of the leading software is ProM [13] (with its ProM Lite variant). In addition, some libraries are becoming popular, since they allow the development of software in different programming languages, such as bupaR [9] (for the R language) and PM4Py [5] (for the Python language). Data can be exchanged between these tools, for example by using the XES standard [7], however, combining these tools together can become cumbersome as it is always necessary to dump the log and change environment.

Another important issue, that only recently is receiving attention [12], is the role of human involvement in the data analysis process [4]. Specifically, all process mining tools currently available do not allow the collaborative execution of mining projects. Many of them provide functionalities to share the results of the analysis, for example via web dashboards, but there is no support for true collaboration and interaction. The increasing availability of network connections and smart-devices should be a key driver of human involvement in mining projects.

To address all these challenges, the tool described in this paper has been designed with the following functional requirements in mind:

– To propose a uniform interface to interact with any number of process mining applications in a seamless way and without the need of confronting every time a steep learning curve;
– To enable collaboration on process mining projects *by design*: not just sharing intermediate or final results, but discussing and performing each individual step of the analysis in real-time and allow all persons involved to contribute to it;
– To enable ubiquitous and truly platform independent execution of process mining projects (e.g., start a mining session at the office PC, continue on the smartphone while commuting, and finish the analysis at home, on a tablet).

The rest of the paper is structured as follows: Sec. 2 reports some technical details about the tool, Sec. 3 describes the main implementation details, and finally Sec. 4 concludes the paper.

2 Technical Details

The strategy we decide to adopt in order fulfill our requirements is to develop a *chat bot* [11] (sometimes also referred to as *chatter bot*) that can be included in a discussion on an instant messaging platform. Such bot, in turn, enables to interact with it in order to perform specific steps of the analysis (i.e., the actual process mining) and sharing the results with all people involved in the discussion.

The general idea behind a chat bot is to exploit the APIs offered by different instant messaging systems and define a software, i.e. the bot, that consumes each message sent by the participants in a discussion. Whenever a message matches a predefined pattern (also known as *command*) which is recognized by the bot, the software performs some actions and sends the response back into the discussion so that all participants can read it. The software that runs the bot has to be executed in a machine, i.e. a server, which has access to the APIs of the instant messaging. In the context of this paper, we decided to use the instant messaging platform Telegram[1] due to its explicit support of chat bots, as well as its portability, which is manifested in a number of implementations available for a large number of platforms, including open source applications[2].

The chat bot we devised, which is called Process Mining Bot (or `pmbot`), assumes that each chat is analyzing a single log at the time. This log, called *the*

[1] See `https://telegram.org/`.
[2] See `https://telegram.org/apps`.

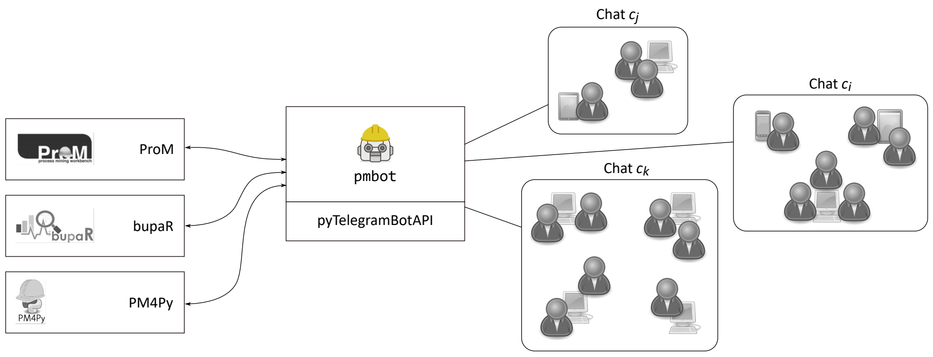

Fig. 1. General architecture of the Process Mining Bot.

current log, is provided to the bot by sharing it within the chat. When a file is shared, the `pmbot` automatically is notified and, if such file contains an event log, the server running the `pmbot` will download it and it will be used for all following analyses. By uploading a new file, the current log is overwritten with the new one. It is important to note that if a file containing the log is uploaded in a chat, it is not necessary for all clients to download it (in fact, only the `pmbot` has to download it). This is of critical importance as it enables the analysis also in devices where memory is a precious resource, such as smart-phones.

As depicted on the right-hand side of Figure 1, the Process Mining Bot can be included in as many chats as needed: it will keep track of each of them individually and therefore each chat will have its own current log. The left-hand side of Figure 1 depicts the three different process mining frameworks, written in three different programming languages that the current implementation of the `pmbot` interacts with ProM [13], bupaR [9], and PM4Py [5]. As previously mentioned, the `pmbot` interacts with the other chat members by "reading" each event being generated in the chat and, whenever one of these messages matches one of the commands, this is recognized and the corresponding action is triggered. The current implementation of the `pmbot` supports the following commands:

- `\start` This command initiates a new process mining session, by asking a license code, needed to use the resources required by the bot;
- `\describelog` Command to produce general log statistics, including the number of traces in the log, the number of activities, their distribution and a couple of charts with the duration of the process instances and the events over time. This command uses the PM4Py library;
- `\hm [DEP_THR]` This command runs the Heuristics Miner algorithm [2] and sends back pictures (one with the heuristics net and another with the Petri net) of the mined model. The default value of the dependency threshold can be overwritten. This command uses the PM4Py library;
- `\alpha` This command performs the Alpha Miner algorithm [3] on the current log. This command uses the PM4Py library;
- `\dfg` This command computes and returns the Direct Following Graph computed on the current log using the PM4Py library;

`\im` Command to perform the Inductive Miner algorithm [10] on the current log and to get the picture of the resulting model. This command uses ProM;

`\precedencematrix` This command produces the precedence matrix of all activities in the current log by using the bupaR library;

`\dottedchart`, `\relativedottedchart` These two commands plot the dotted chart of the current log by using either a relative or absolute time scale. These commands use the bupaR library;

`\resources` Command to plot how often each resource is involved in an event, by using the bupaR library;

`\keepactivities` This command changes the current log by filtering it in order to keep only some activities. The user can enter the name of all activities to keep until a special termination command is given;

`\removefilters` This command can be used to reset the current log to its original form (i.e., the same status as it has been uploaded). It is useful to reset the current log whenever the users want to the applied filters.

The current set of commands already allows several interesting analyses. Actually, in all data and process mining projects it is arguable that the most important element of the analysis are the users who have to be smart in *(i)* understanding which algorithm is the most suitable in the given setting and *(ii)* interpreting the results of the mining algorithm. On top of this, the ability to recursively apply filters allows the investigation to go beyond the raw log.

Whenever treating data, one of the most delicate elements is the confidentiality of the data. Whenever implementing a bot it is important to understand that it is actually just an application running on someone's machine and the owner of such machine has access to the log. Therefore, in case the data to be mined is confidential it might be wise to have an *in house* and trusted machine allocated to the purpose. In addition to the machine running the `pmbot`, when the log is uploaded it has to transit over the Telegram's servers. To impede Telegram from being able to read the log, the current implementation supports encrypted zip files for transmitting the file. Of course, more advanced techniques (e.g., GPG) might be easily implemented as well.

A second element to be considered is the accountability of the activities performed by the bot: if the log file is very large, the machine running the `pmbot` might require a vast amount of resources. For this reason, in the current implementation of the bot we introduced a limit on the size of the event log (easily adjustable in the configuration file) and, in order to send any command, it is necessary to enter a "license code". This code enables the access to specific set of functionalities and time-based coded might also be implemented to grant access only for limited periods.

3 Implementation Details

The Process Mining Bot has been implemented as a standalone Python software, exploiting the Telegram API[3] and, in particular, the corresponding Python

[3] See `https://core.telegram.org/bots/api`.

wrapping implemented in the pyTelegramBotAPI[4]. The current version of the software embeds the PM4Py libraries. In order to execute bupaR and ProM, external calls are done using the `subprocess` Python module. The complete source code of the bot is available in GitHub at `https://github.com/delas/pmbot`. A screencast showing all functionalities in a mock-up session is available at `https://youtu.be/eg8jJ3bBONI`. Finally, a detailed tutorial presenting a walk through of the basic steps in order to interact with the bot is available at [6].

4 Conclusions

In this paper we presented an innovative technique of completing process mining projects in a collaborative, ubiquitous, platform independent and integrated manner. The technique exploits the idea of chat bots and, by including them into existing instant messaging discussions, it is possible to have direct access to process mining functionalities. The prototype implementation presented in this paper is available for Telegram and the bot is capable of executing commands on ProM, bupaR, and PM4Py.

References

1. van der Aalst, W.M.: Process Mining. Springer, second edn. (2016)
2. van der Aalst, W.M., Weijters, T.A.J.M.M.: Rediscovering Workflow Models from Event-based Data Using Little Thumb. Integr Comput Aided Eng **10**(2) (2003)
3. van der Aalst, W.M., Weijters, T.A.J.M.M., Maruster, L.: Workflow Mining: Discovering Process Models from Event Logs. IEEE Trans Knowl Data Eng **16** (2004)
4. Ankerst, M.: Human Involvement and Interactivity of the Next Generation's Data Mining Tools. In: ACM SIGMOD DMKD, position paper (2001)
5. Berti, A., van Zelst, S., van der Aalst, W.: Process Mining for Python (PM4Py): Bridging the Gap Between Process- and Data Science. In: CEUR-WS Online Proceedings of the ICPM Demo Track (2019)
6. Burattin, A.: `pmbot` Tutorial (2019), `https://github.com/delas/pmbot/wiki`
7. Günther, C.W., Verbeek, E.H.M.W.: XES Standard Definition. www.xes-standard.org (2009), `http://www.xes-standard.org/`
8. IEEE Task Force on Process Mining: Process Mining Manifesto. In: Business Process Management Workshops. pp. 169–194. Springer-Verlag (2011)
9. Janssenswillen, G., Depaire, B., Swennen, M., Jans, M., Vanhoof, K.: bupaR: Enabling reproducible business process analysis. Knowl Based Syst **163** (2019)
10. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.: Discovering Block-Structured Process Models from Event Logs - A Constructive Approach. In: Proceedings of Petri Nets. pp. 311–329. Springer Berlin Heidelberg (2013)
11. Mauldin, M.L.: ChatterBots, Tinymuds, and the Turing Test: Entering the Loebner Prize Competition. In: AAAI. vol. 94, pp. 16–21 (1994)
12. Minku, L.L., Mendes, E., Turhan, B.: Data mining for software engineering and humans in the loop. Progress in Artificial Intelligence **5**(4), 307–314 (Nov 2016)
13. Verbeek, E.H.M.W., Buijs, J., van Dongen, B., van der Aalst, W.M.: ProM 6: The Process Mining Toolkit. In: BPM 2010 Demo. pp. 34–39 (2010)

[4] See `https://github.com/eternnoir/pyTelegramBotAPI`.