Rijksuniversiteit Groningen
Computing Science | Thesis Project

A Tool for Log Generation of Adaptive Business Processes

Author:

Dyllan Cartwright d.p.cartwright@student.rug.nl

Supervisors:

Arash Yadegari a.yadegari.ghahderijani@rug.nl
Dimka Karastoyanova d.karastoyanova@rug.nl

Collaborator:

Radu Andrei Sterie r.a.sterie@student.rug.nl

30 July 2023

Contents

Acknowledgments

I would like to express my sincere gratitude to all those who have supported me throughout the process of completing this thesis.

First and foremost, I am immensely grateful to my supervisor, Arash Yadegari, for their invaluable guidance, expertise, and continuous encouragement. Their insightful feedback and constructive suggestions have been instrumental in shaping the direction of this work. But perhaps, I am most grateful for their genuine kindness and for them (*willingly*) engaging in meetings that sometimes took 2+ hours.

Additionally, I would like to acknowledge the contributions/collaborations made with my colleague (and one of my closest friends) Radu Andrei Sterie.

I would also like to extend my thanks to Kuljit Dhami (another one of my closest friends), for her invaluable assistance in proofreading my thesis. Her literary skills are unrivalled (her words) and have been hugely helpful.

Furthermore, I extend my gratitude to the Rijksuniversiteit Groningen itself, for providing me with the opportunity to pursue my studies and for fostering an environment of learning and growth. I am grateful for the knowledge and skills I have gained during my time here.

Finally, I would like to thank my mum for her unwavering support, understanding, and patience throughout my journey in this city. Her encouragement and belief in me has been a constant source of motivation. I truly hope she knows she has been the backbone and inspiration to me completing my education.

Since I can remember, my mum has been the fuel for me to better myself - I hope with all of my heart to repay her by improving not only my own future but also hers.

Abstract

This thesis project focuses on the development of >_next(log), a sophisticated tool for generating logs to be used with adaptive business processes.

The tool aims to provide a user-friendly and intuitive interface, streamlining the process of log generation for adaptive processes. It minimises the need for manual intervention or input by incorporating features such as a graphical user interface for creating a "Rule List", whereby users can adapt business logs by using their own custom rules. The ultimate goal is to automate the log generation process and enhance the usability of the application.

Drawing inspiration from previous research done by my supervisors Arash Yadegari and Dimka Karastoyanova [1, 2], this thesis leverages an approach for synthetic log generation that focuses on control-flow changes in KPI-based process adaptations. By allowing users to upload business process logs in .mxml format and corresponding business process models in .bpmn format, >_next(log) empowers users to view the uploaded models within the application. The tool's most significant feature is the ability for users to define precise sets of "Rules", which can then adapt the provided logs accordingly.

The generated logs can then be used in conjunction with the ml.log thesis project by Radu Andrei Sterie to predict and identify (the injected) patterns in the logs using machine learning techniques [3].

With the use of >_next(log), users can overcome the challenge of limited access to real-world business data sets, enabling more comprehensive analysis and understanding of business processes.

Keywords: Process adaptation, Process change, Synthetic data generator, Event log generation, Business process simulation.

1 Introduction

Business Process Management (BPM) is a multidisciplinary research field that centers around the modeling, implementation, analysis, and improvement of business processes. It encompasses various domains and disciplines to effectively manage and optimize organizational workflows and operations [4, 5]. A significant part of this research relies on data generated by Business Process Management Systems (BPMS). However, due to the proprietary nature of business processes, companies are often reluctant to expose their process data, leading to a shortage of data for research purposes. This thesis introduces >_next(log), a tool designed to generate synthetic event logs for adaptive business processes, thereby providing a solution to this data shortage.

1.1 Background and Scope

Event logs are essential for analysing data generated by BPMS. These logs comprise a collection of process traces, where each trace represents a sequence of events. Each event within a trace contains relevant information, such as the activity name, activity timestamp, and other attributes (e.g. cost) [6].

Business Process Simulation plays a crucial role in Business Process Management (BPM) as it enables the analysis and improvement of defined business processes. This powerful technique offers a wide range of applications, including optimising customer service, enhancing production lines, and identifying opportunities for cost savings. By simulating business processes, organisations can gain valuable insights to drive efficiency, make informed decisions, and continuously enhance their operations [7]. However, the traditional approach to Business Process Management, which often involves manual intervention and input, can be time-consuming and prone to errors. This is particularly true when it comes to log generation for adaptive processes [8, 9].

The motivation behind the development of >_next(log) is to address these challenges by automating the log generation process and enhancing the usability of the application. By doing so, >_next(log) aims to streamline the process of log generation for adaptive processes, thereby contributing to the field of business process analysis.

The primary objective of this thesis is to present >_next(log), a tool designed to generate logs for adaptive business processes. The tool leverages an approach for synthetic log generation that focuses on control-flow changes in KPI-based process adaptations. By allowing users to define precise sets of "Rules", >_next(log) enables the adaptation of provided logs according to the given rules and BPM.

Furthermore, this thesis aims to showcase the practical application of the software >_next(log) in conjunction with machine learning techniques to discover causal rules in the adapted logs, particularly with the ml.log thesis project by Radu Andrei Sterie. Previous research has already demonstrated the potential of machine learning in this context [2, 10, 11]. ml.log will use the generated logs from >_next(log) to attempt to predict and identify (the injected) patterns.

The scope of this thesis includes the presentation of the architecture and technologies of >_next(log), the definition of its "Rules" grammar, and the evaluation of the tool's effectiveness in generating adaptive logs for business processes.

1.2 Related Work

1.2.1 Adaptive Business Processes

Adaptive business processes are a critical aspect of modern business operations, enabling organisations to respond swiftly and effectively to changes in their operating environment. These processes are designed to be flexible and can be modified in real-time to meet evolving business needs [12].

1.2.2 Log Generation and Synthetic Data

Log generation and synthetic data play a crucial role in the analysis and improvement of business processes. Event logs are typically generated by Business Process Management Systems (BPMS) and contain valuable information about the execution of business processes. Each event log consists of a set of process traces, with each trace representing a sequence of events. The data attached to each event can include the activity name, activity timestamp, and other event-related attributes, such as cost [6].

Synthetic data, on the other hand, refers to data that is artificially generated

rather than collected from real-world events. In the field of BPM, synthetic event logs can be used to simulate the execution of business processes, allowing for the testing and validation of different process models. As stated before, this can be particularly useful in situations where real-world data is scarce or sensitive.

1.2.3 Existing Tools

Several papers have explored the generation of artificial event logs, although mostly for discovering BPMNs from logs, they are still relevant;

1. One approach involves manually creating logs, but this method is time-consuming and prone to errors [13].
2. Another approach utilises CPN IDE (formerly known as CPN Tools), a Petri nets editor with simulation capabilities, to generate random logs based on a given Petri net model. However, this approach requires writing scripts in Standard ML, which can be challenging [14].
3. Transforming BPMN models to DEVS formalism and simulating them using DEVS simulation tools is yet another approach, but it involves additional steps and transformations [15, 16, 17, 18].
4. Other tools such as SecSy focus on generating event logs for security-oriented information systems, while PLG2 offers configurable toolboxes for event data generation based on artificial business process models [19, 20, 21]. PLG2 also allows for random evolution of a process model, in order to generate a slightly different version of it (to simulate concept drifts).
5. BPMN engines like Bizagi (which is considered to be one of the best BPM tools by users [22]) can also generate event logs, however it is not free.
6. Another option is BonitaSoft (with an available open-source edition), which offers graphical representation of business processes using BPMN. While its intuitive drag-and-drop modeling interface simplifies process design, the tool's simulation capabilities are still quite bare, and do not allow for many probability distributions. [22]

7. Finally, there is BIMP, which is a free online simulator that offers simplicity through adjustable parameters and probability distributions when creating simulations. Although it requires a separate modeling tool to develop the business process model, BIMP allows users to create fairly customisable simulations (with some limitations).

While all of these tools offer the capability to generate synthetic logs by defining probability distributions and other parameters, none[1] of them seem to provide the desired feature that allows users to dynamically adapt the business process during a trace, solely based on user-defined rules applicable to that *specific* trace. As will be seen later on, these rules are built upon the trace's KPI metrics (eg `duration`, `cost` etc).

Given that the tools listed above are available and are good enough for creating unadapted event logs, >_next(log) takes a different approach by serving as a complementary module in the pipeline. Rather than reinventing the wheel, it focuses on enhancing the subsequent step. Hence, users can generate their initial logs using any preferred method[2] and then >_next(log) becomes a tool for adapting these (initial) logs according to user-defined rules.

While in theory, any generative tool should work, BIMP was chosen as the primary way to generate logs for this thesis project. Although BIMP has some implementation limitations and requires additional data processing for specific use cases, it is straightforward to generate initial event logs. This makes it a solid starting point for us to create a prototype of >_next(log).

Likewise, the bpmn.io tool will be used in conjuntion with BIMP to create the BPMNs.

1.3 Objective

Hence, the overarching goal of >_next(log) is to provide users with the flexibility to generate their initial logs using any preferred method and subsequently adapt these logs based on user-defined rules, whilst ensuring an intuitive and coherent UI.

[1] Bizagi does have a "what-if" analysis feature, which may implement the desired feature, however, because I don't have access to it, I can't be sure.

[2] Though, as will be seen in the following sections, the initial logs will need to be generated in MXML format.

2 Methodology

This section delves into the approach and techniques employed in the development of >_next(log). It begins with a diagram depicting >_next(log)'s pipeline to generate adapted logs. The supported data formats are discussed, followed by an exploration of the technologies utilised. The architecture of >_next(log) is briefly touched upon. Next we cover the usage of >_next(log), including rule definitions and examples of its implementation. Finally, the limitations and assertions of the tool are discussed.

2.1 Pipeline

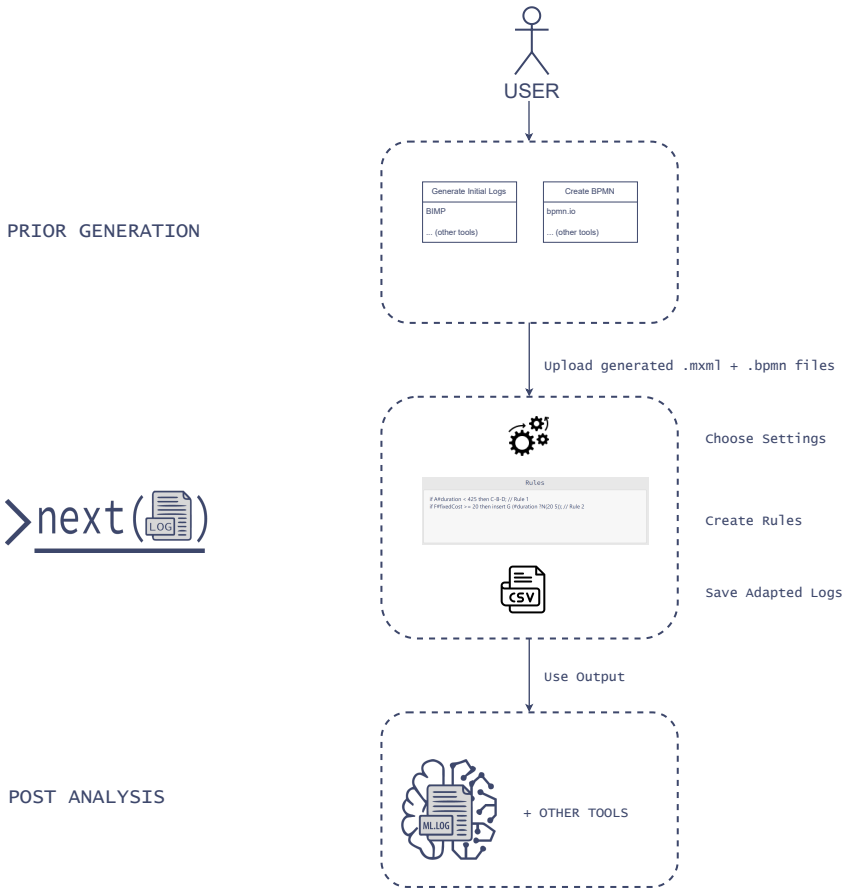


Figure 1: The Flow/Pipeline of >_next(log)

2.1.1 Data Formats

MXML

The MXML (Mining eXtensible Markup Language) data format is widely used for representing event logs in the field of process mining. MXML provides a standardised way to capture and store event data related to business processes. It includes information such as the process activities, timestamps, and other relevant attributes. These logs will be used by >_next(log) as the unadapted logs.

Users can (in theory) generate synthetic MXML files however they want and it should work with >_next(log), but >_next(log) has only been tested on logs generated by the BIMP tool.

BPMN

On the other hand, the BPMN (Business Process Model Notation) data format is designed specifically for modeling business processes. BPMN diagrams provide a standardised and intuitive way to depict the sequence of activities, decisions, gateways, and other elements within a business process model (as XML data).

>_next(log) utilises the uploaded BPMN file to establish correlations between the logs and the defined "Rules" while also performing basic error checking. Likewise, >_next(log) is expected to support BPMN files generated anywhere, but has only been tested using BPMN files made by the bpmn.io tool.

2.2 Technologies of >_next(log) (& ml.log)

Python was chosen as the backend programming language. Python is a versatile and widely-used programming language known for its simplicity and readability. It offers several advantages that made it a suitable choice for developing the backend of >_next(log):

- **Intuitive Development**: Python's clean and expressive syntax makes it intuitive to work with, allowing for efficient and straightforward development.
- **Extensive Library Ecosystem**: Python has a vast collection of

open-source libraries and frameworks, which played a significant role in the decision to use it for >_next(log). Those libraries provided ready-made solutions for various tasks, such as data processing, "Rule" lexing, machine learning techniques found in ml.log and finally, seamless frontend development.

- **Supervisor's Research**: Arash Yadegari's previous work was also done in Python.

The frontend of >_next(log) was made with PySide6, a Python binding for the Qt framework.

The decision to use PySide6 for the UI in >_next(log) was based on several factors:

- **Customisability**: PySide6 offers a high level of customisation, allowing for fine-grained control over UI elements and enabling the creation of a tailored and cohesive user interface for >_next(log).

  This was the most appealing factor for me, I wanted a smooth and engaging user experience whilst defining "Rules" through the UI - and PySide6 made this possible.

- **Cross-platform Compatibility**: Qt, the underlying framework of PySide6, supports cross-platform development, ensuring that the UI created with PySide6 can run smoothly on different operating systems, including Windows, macOS, and Linux.

  This was also very important, as we wanted one codebase that worked (and looked the same) for any OS.

- **Documentation**: PySide6's documentation is exceptional and provided an enjoyable experience during its utilisation. As you may agree, finding documentation that is enjoyable to read is certainly an uncommon occurrence.

>_next(log) also used the PLY (Python Lex-Yacc) library. PLY is a Python implementation of the well-known Lex and Yacc tools, commonly used for building compilers and parsing tools.

Its benefits included:

- **Legal Rule Definition**: Most importantly, PLY facilitated the definition

of >_next(log)'s formal grammar and hence, the enforcement of properly structured rules.

- **Error Handling**: PLY includes built-in error handling mechanisms, providing informative error messages when encountering syntax errors or rule violations in the input data.

Finally, the xml library in Python was utilised in >_next(log) to parse both the MXML and BPMN files.

Note: The libraries, "matplotlib, pandas, scikit-learn, sklvq" were also used in ml.log, as well as the software graphviz.

2.3 Architecture of >_next(log) (& ml.log)

Our codebase follows a well-structured architecture that is both intuitive and scalable. It is organised into two main directories: "frontend" and "backend".

In the "frontend" directory, you will find the code responsible for defining the user interface (UI) using QML (Qt Meta-Object Language). QML is a declarative language that allows the creation of visually appealing and interactive UI components.

Both within the "frontend" and "backend" directories, the code is further divided into "log" and "ml" subdirectories.

Hence, the "frontend/log" directory contains the QML code specific to >_next(log), focusing on the UI components and functionalities related to log generation, adaptation, and rule definition. On the other hand, the "frontend/ml" directory encompasses the QML code specifically related to ml.log.

Moving to the "backend" directory, you will find the "controller.py" file, which serves as the API between the frontend UI and the underlying functionality of >_next(log). This file acts as a bridge, facilitating communication and data exchange between the frontend UI and the backend logic.

The structure of >_next(log)'s codebase is designed to be intuitive and scalable, allowing for easy maintenance, expansion, and modular development. This architectural approach separates the concerns of the UI and the backend functionality, as well as logic related to >_next(log) and ml.log, thus ensuring a clear separation of responsibilities.

2.4 Using >_next(log)

This section provides a quick guide on effectively utilising >_next(log) for generating and adapting business process logs. The following will walk you through the various pages and functionalities within the software.

You can see screenshots of >_next(log) in the appendix: A.

- **Home page**: On this page, users can easily upload their .bpmn and .mxml files. This page serves as the starting point for the log generation process, allowing users to select the necessary files for adaptation.
- **View page**: This page provides users with a visual representation of their uploaded BPM. It enables users to inspect and review the structure and flow of the process model, aiding in the understanding of the process when creating rules.
- **Rules Editor page**: On this page, users can create their rules for log adaptation. The intuitive interface allows users to define rules based on specific conditions and actions. After creating the rules, users can click 'Parse Rules' to verify their validity. By clicking 'Available', users can view all the defined events and attributes to facilitate rule creation.
- **Save page**: This page is where users can generate and save the adapted logs. Users can specify the desired output file name and save the adapted logs as a .csv file. Additionally, users have the option to save the "original" unadapted logs as a separate .csv file, providing traceability and allowing for debugging purposes if needed.
- **Settings page**: This page offers users the ability to customise certain preferences according to their requirements. Users can define the default folder location for file uploads. If the default folder location setting is disabled, >_next(log) will remember the last folder used for uploading files and will open there.
- **The Switch Icon**: The Switch Icon allows users to toggle between >_next(log) and ml.log, another tool within the software. This feature provides users with flexibility and the ability to easily switch between the two applications.

2.4.1 Bonuses

One of the main goals of this project was that it must be user-friendly, intuitive, and should provide a seamless automated process for generating logs for adaptive processes. Hence, the application was designed to streamline this process and minimise the need for manual intervention or input. >_next(log) not only meets but also exceeds these expectations for several reasons:

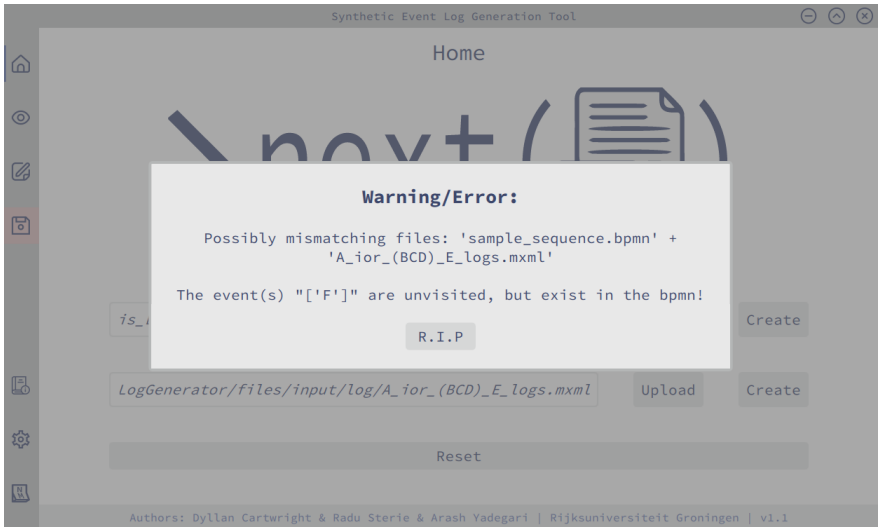
1. **Easy file uploading**: Uploading .mxml and .bpmn files is a straightforward process that only requires a few clicks.

   Additionally, the tool provides convenient buttons that quickly link users to the BIMP and bpmn.io tools, simplifying the file selection and uploading process.

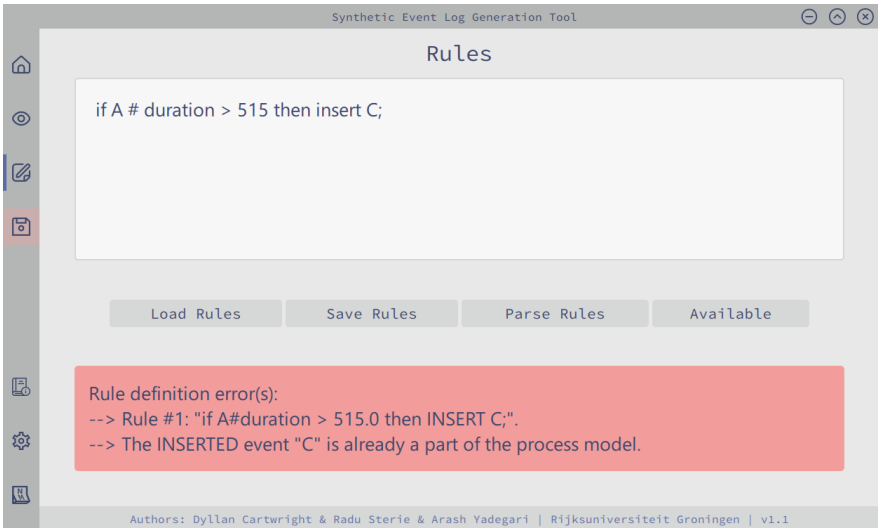
2. **Built-in BPM Viewer**: >_next(log) includes a built-in BPM viewer, allowing users to effortlessly check the uploaded BPM when generating rules. This feature enhances usability and provides users with a visual representation of the process model.

3. **Extensive error checking**: >_next(log) incorporates error checking mechanisms;

   - For instance, when uploading files, the tool performs compatibility checks to ensure the files match. If any inconsistencies are detected, helpful warnings are displayed to guide users (as seen below).



- Similarly, the parser in >_NEXT(LOG) is adept and robust at verifying rule accuracy, providing informative messages when users make errors. This proactive error checking enhances the user experience and helps users avoid mistakes.



4. **Expanded functionality**: >_NEXT(LOG) has expanded its functionality significantly beyond the initial project description;

   - Originally, the tool only required the `skip` and `insert` actions. However, it now includes additional actions such as `+` (to parallel action) and `-` (to series action).
   - Moreover, time shifting capability has been added (with preserved waiting times).
   - Initially, "Cycle Time" and event durations were the only expected attributes available to create rules with, but now users can utilise any available attribute. Likewise, multiple quantifiers, including `#`, `@`, and `!`, have been incorporated to further enhance flexibility in rule creation.

     Note: to create a rule using an instance's "Cycle Time", you could use the following; `if _End@duration ... then ...`.

5. **Exemplary UI**: The user interface (UI) of >_NEXT(LOG) probably surpasses expectations. Its intuitive design and user-friendly layout

contribute to a positive user experience. The UI was crafted to provide users with easy navigation, clear instructions, and an overall efficient workflow.

2.4.2 Output

The output of >_next(log) is a .csv file that includes the following components;

1. **Attribute values**: The file has column headers that list all the events present in the business process model and their associated attributes. These headers facilitate easy identification and understanding of the events and their corresponding attributes.
2. **Instance path**: The .csv file describes the path taken by each instance within the business process, i.e., it outlines the sequence of events followed by each (possibly adapted) instance.
3. **Rule definitions**: For each user-defined rule, a corresponding column is included in the .csv file. The column name represents the rule definition. Each cell in the rule column indicates whether the rule was applied or not for a particular instance. If the rule was applied, then the cell value is 1; otherwise, it is 0.

   It's important to note that even if a rule's condition evaluates to true, it does not guarantee that the rule was applied. For example, in BPM 3, if a specific instance's path was '_Start->A-> C->_End' and the rule stated `if A#duration > 515 then skip B;` the rule column would still have a value of 0, regardless of A's duration exceeding 515.
4. **"_Label" column**: The "_Label" column in the .csv file represents a binary representation of all possible combinations of applied rules for each instance. This column provides a concise summary of the rules that were actually applied to the instances, allowing for easy identification of patterns and correlations between different rules.

   Likewise, this allows for seamless integration into ML.log.
5. **Save original:** Finally, >_next(log) also provides the option for users to save the original (unadapted) logs, primarily for debugging and traceability purposes.

2.5 Rule Definitions

In this section, we present the formal grammar and syntax for defining rules within >_next(log) to facilitate the adaptation of business process logs.

These rules are expressed in a Backus-Naur Form (BNF) grammar, providing a structured and consistent framework for specifying log adaptation criteria. We will explain the different tokens, terminals and quantifiers used in the grammar, enabling users to understand and construct their own custom rules effectively.

2.5.1 Grammar

⟨*rules*⟩ ::= ⟨*rules*⟩ ⟨*rule*⟩ '`;`' ⟨*comment*⟩?
| ⟨ϵ⟩ ⟨*comment*⟩?

⟨*rule*⟩ ::= '`if`' ⟨*identifier*⟩ ⟨*expr*⟩ '`then`' ⟨*action*⟩
| '`if`' ⟨*rt_expr*⟩ '`then`' ⟨*action*⟩

⟨*expr*⟩ ::= '`#`' ⟨*identifier*⟩ ⟨*equality*⟩ ⟨*value*⟩
| '`@`' ⟨*identifier*⟩ ⟨*equality*⟩ ⟨*value*⟩

⟨*rt_expr*⟩ ::= '`!`' ⟨*identifier*⟩ ⟨*equality*⟩ ⟨*value*⟩

⟨*action*⟩ ::= '`skip`' ⟨*identifier*⟩
| '`insert`' ⟨*identifier*⟩ '`(`' ⟨*attributes*⟩ '`)`'
| '`insert`' ⟨*identifier*⟩
| ⟨*parallel_list*⟩
| ⟨*sequential_list*⟩

⟨*parallel_list*⟩ ::= ⟨*parallel_item*⟩
| ⟨*parallel_next*⟩

⟨*parallel_item*⟩ ::= ⟨*identifier*⟩ '`+`' ⟨*identifier*⟩

⟨*parallel_next*⟩ ::= '`+`' ⟨*identifier*⟩
| ⟨*parallel_next*⟩ '`+`' ⟨*identifier*⟩

⟨*seq_list*⟩ ::= ⟨*seq_item*⟩
| ⟨*seq_next*⟩

⟨*seq_item*⟩ ::= ⟨*identifier*⟩ '`-`' ⟨*identifier*⟩

⟨*seq_next*⟩ ::= '-' ⟨*identifier*⟩
| ⟨*seq_next*⟩ '-' ⟨*identifier*⟩

⟨*attributes*⟩ ::= ⟨*attributes*⟩ '#' ⟨*identifier*⟩ ⟨*value*⟩
| ⟨*attributes*⟩ '#' ⟨*identifier*⟩ ⟨*distribution*⟩
| '#' ⟨*identifier*⟩ ⟨*value*⟩
| '#' ⟨*identifier*⟩ ⟨*distribution*⟩

⟨*distribution*⟩ ::= '?N' '[' ⟨*value*⟩ ⟨*value*⟩ ']'
| '?U' '[' ⟨*value*⟩ ⟨*value*⟩ ⟨*value*⟩ ']'

⟨*identifier*⟩ ::= [a-zA-Z_][a-zA-Z0-9_]*

⟨*value*⟩ ::= [-]?[0-9]+['.'[0-9]+]?

⟨*equality*⟩ ::= '<=' | '>=' | '<' | '>' | '==' | '!='

⟨*comment*⟩ ::= '//' [^\n]*

⟨ϵ⟩ ::= ''

2.5.2 Quantifiers

- The '#' (THIS_TOK) is a quantifier which means *this* event's attribute value. Hence, `A#duration` means the duration of A.
- The '@' (ACCUMULATIVE_TOK) is a quantifier which means the accumulative attribute value *at* that event (inclusive).

  Hence, `C@duration` means the total duration of the instance after event C.
- The '!' (RUNNING_TOTAL_TOK) is a quantifier which means the total running attribute value during the instance. This

  quantifier does not have a event identifier attached to it.

  Hence, `if !duration > 15 then skip D;` means that if at *any* point during the instance, if the running total for duration is > than 15, then `skip` D. Note: if D has already happened by the time the condition becomes true, then the rule is ignored.

2.5.3 Keywords / Actions

For the more intricate details, read the "Examples/Implementations" section 2.6 further below.

- The keywords `if` and `then` are self-explanatory.
- `skip`
  - This action will skip an event that otherwise *would* have happened.

    Hence, imagine `if A#duration > 15 then skip B;` with BPM 3 (shown below). If an instance's `A#duration` was greater than 15, then the rule will only be 'applied' if the instance was going to choose 'B' in the xor gate. If the instance took the path "_Start->A->C->_End" then the rule will not be applied regardless of A's duration.
  - Proceeding events' timestamps will be updated accordingly, they will be 'shifted' to the left by the skipped event's duration. Waiting times will be preserved.
  - If the skipped event was part of a parallel gateway, then the proceeding events' timestamps will be updated accordingly *if* the skipped event was the last event to finish in the gateway, otherwise nothing is changed - of course, this is different per instance.
- `insert`
  - This action will insert the given event *right after* the event that triggered it.

    Hence, `if A#duration > 515 then insert D;` will insert D right after A. If A was in a parallel gateway, then "A->D" is now part of the same gateway.
  - Proceeding events' timestamps will be updated accordingly, they will be 'shifted' to the right by the inserted event's duration. Waiting times will be preserved.
  - If the inserted event was part of a parallel gateway, then the proceeding events' timestamps will be updated accordingly *if* the

inserted event was the last event to finish in the gateway (for that instance).

- `+ // TO_PARALLEL_TOK`
  - This action means for all the events in the given chain, turn them into parallel events.

    Hence, `if A#resourceCost > 10 then B+C+D;` means make the events B, C, and D parallel.
  - The chain needs to be such that the events are (directly) in series. For example, looking at BPM 2 below, `... B+C` is allowed, so is `... B+C+D` but `... C+B` and `... B+D` is not.
- `- // TO_SERIES_TOK`
  - This action means for all the events in the given chain, turn them into events in series.

    Hence, `if A#resourceCost > 10 then B-C-D;` means make the events B, C, and D into series. Note: order is important, ie; if you specify `... then C-D-B` then that will be the order taken, where waiting time will be preserved.
  - Likewise, you can only put events in series if they are in parallel, and *all* events must be transformed to series. For example, looking at BPM 5, `... D-E-F` is allowed, but `.. D-E` is not (because F is missing).

2.5.4 Terminals

- <rules> is your list of rules separated by the ';' token.
- <distribution> is a terminal which allows a user to define an attribute distribution type when inserting an event, the available ones are;
  - `?N (mu sigma)`: A normal distribution, with mean μ and standard deviation σ.
  - `?U (min_val max_val step_size)`: A uniform distribution with the given parameters.

  - Likewise, there is also a "Fixed Distribution" type, but is not actually a <distribution> terminal.
  - For example, if you are inserting the event B, with the attributes a1, a2, a3, a4, a5, you could write the rule;

    ```
    .. insert B (#a1 ?N(20 3) #a2 ?U(10 20 0.2) #a3 10);
    ```

    This means B will be inserted with attributes a1 following a normal distribution, a2 with a uniform distribution, a3 will always be 10, and finally a4 and a5 will be set to 0.

    Do note; the attribute `#occurred` for the inserted event will always be set to 1 if the event was inserted (otherwise it will be 0). Likewise, `#start_time` and `#end_time` will be created dynamically (with `#start_time` being right after the previous event's `#end_time`). You do not have to do this yourself, these will happen automatically.

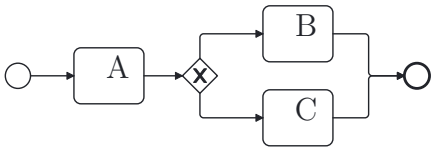
- <identifier> is any identifier using the letters from the Roman alphabet, underscores and numbers. It must start with a letter or underscore. These are used for event names and for attribute names.
- <value> is any number, float or integer (positive or negative).
- <comment> is a normal comment, everything after '//' is ignored (until a newline).

2.6 Examples / Implementations

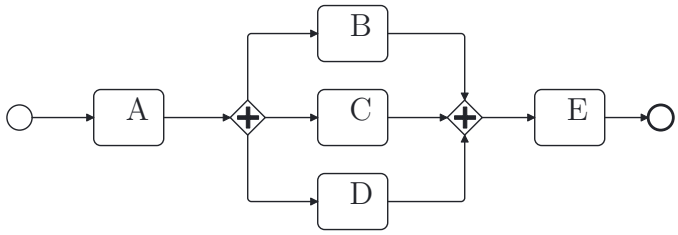
Suppose we had the following BPMNs;

BPM 2

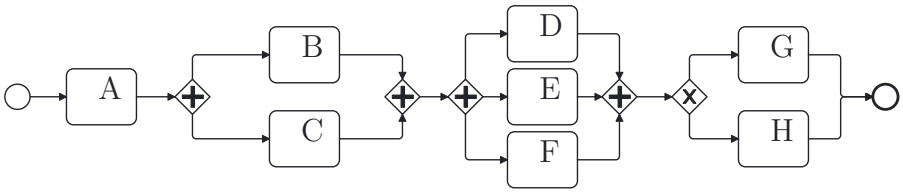
BPM 3



BPM 4



BPM 5



And suppose we had the following rules, where the comments `... // A:1,3; R:2,4` mean that the >_next(log) parser accepted the rule for BPMs 1 and 3, and rejected the rule for BPMs 2 and 4;

```
1 if A#duration > 515 then skip B;     // A:1,2,3,4
2 if B#duration > 515 then insert K;   // A:1,2,3,4
3 if A#duration > 515 then insert D;   // A:2; R:1,3,4
4 if A#duration > 515 then B+C;        // A:1; R:2,3,4
5 if A#duration > 515 then C+B;        // R:1,2,3,4
6 if A#duration > 515 then B-C;        // A:4; R:1,2,3
7 if A#duration > 515 then C-B;        // A:4; R:1,2,3
8 if A#duration > 515 then D-F-E;      // A:4; R:1,2,3
```

```
9  if A#duration > 515 then D-F;          // R:1,2,3,4
10 if !duration > 1000 then B-C;          // A:4; R:1,2,3
```

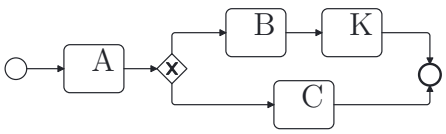
1: `if A#duration > 515 then skip B;  // A:1,2,3,4`

- BPMs 2; the output works as expected. If the rule's condition evaluates to true for that instance, then B will be skipped and all of the proceeding event's timestamps will be shifted accordingly, where waiting times are preserved (you can assume waiting times are always preserved).
- BPMs 3; the rule will only be applied if the instance was going to 'choose' B at the xor gate, otherwise even if A's duration > 15, then nothing will happen.
- BPMs 4,5; B will be skipped if the rule's condition evaluates to true. And if B was the last event to finish in the parallel gateways, then the proceeding event's timestamps will be shifted to the left (otherwise nothing happens to their timestamps). Of course, this may be different per instance.

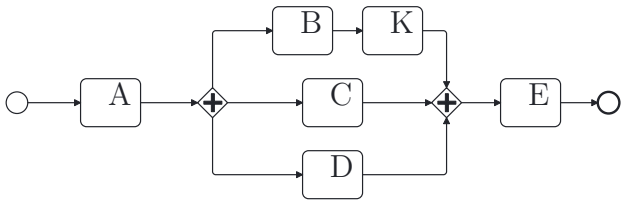
2: `if B#duration > 515 then insert K;  // A:1,2,3,4`

- BPMs 2; the output works as expected. If the rule's condition evaluates to true for that instance, then K will be inserted right after B and all of the proceeding event's timestamps will be shifted accordingly.
- BPMs 3; the rule will only be evaluated if the instance 'chose' B at the xor gate, otherwise the rule isn't even evaluated. Likewise, if the instance chose B, and B's duration was > 15 then K is inserted right after B, with all the proceeding events' timestamps shifted accordingly (see BPM 6 below).
- BPMs 4,5; K will be inserted after B (in series *but* still in the parallel gateway, see BPM 7 below) if the rule's condition evaluates to true. And if K was the last event to finish in the parallel gateway, then the proceeding event's timestamps will be shifted to the right (otherwise nothing happens to their timestamps). Of course, this may be different per instance.

BPM 6



BPM 7



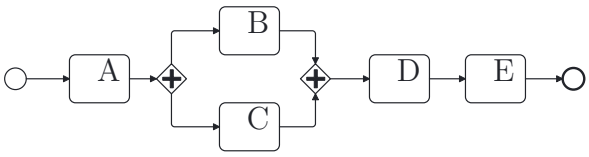
3: `if A#duration > 515 then insert D;  // A:2; R:1,3,4`

- BPMs 3; the output works as expected and works as explained above.
- BPMs 2,4,5; the >_NEXT(LOG) parser will reject this rule as D already exists in th BPM.

4: `if A#duration > 515 then B+C;  // A:1; R:2,3,4`

- BPMs 2; The output works as expected, if the rule's condition evaluates to true, then the instance's BPM will look like BPM 8 seen below. Of course, the proceeding event's timestamps will be shift to the left accordingly (with waiting time preserved)
- BPMs 3,4,5; the >_NEXT(LOG) parser will reject this rule as events B and C are not (directly) in series with each other.

BPM 8



5: `if A#duration > 515 then C+B;  // R:1,2,3,4`

- BPMs 2,3,4,5; the >_NEXT(LOG) parser will reject this rule as events B and C are not (directly) in series with each other.

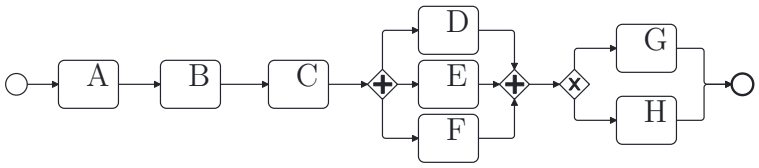
6: `if A#duration > 515 then B-C;  // A:4; R:1,2,3`

- BPMs 5; The output works as expected, if the rule's condition evaluates to true, then the instance's BPM will look like BPM 9

seen below. Of course, the proceeding event's timestamps will be shift to the right accordingly (with waiting time preserved)

- BPMs 2,3,4; the >_NEXT(LOG) parser will reject this rule as events B and C are not in parallel with each other - note how for BPM 4, the event D is not in the chain.

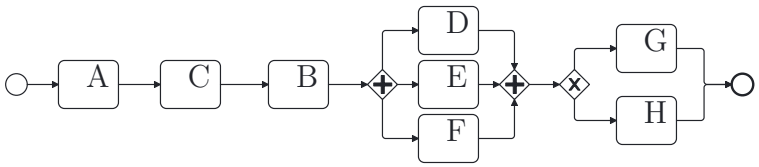
BPM 9



7:
```
if A#duration > 515 then C-B;          // A:4; R:1,2,3
```

- BPMs 5; Like before, the output works as expected. However, note the order. If the rule's condition evaluates to true, then the instance's BPM will look like BPM 10 seen below. The original start_time of C is kept and everything is shifted accordingly.
- BPMs 2,3,4; the >_NEXT(LOG) parser will reject this rule as events B and C are not in parallel with each other - note how for BPM 4, the event D is not in the chain.

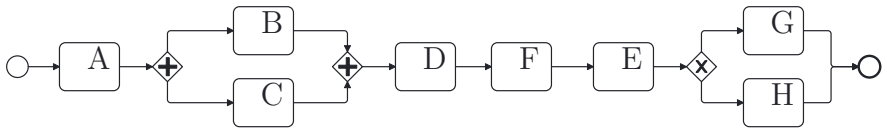
BPM 10



8:
```
if A#duration > 515 then D-F-E;        // A:4; R:1,2,3
```

- BPMs 5; Like before, the output works as expected. If the rule's condition evaluates to true, then the instance's BPM will look like BPM 11 seen below.

BPM 11



9: `if A#duration > 515 then D-F;  // R:1,2,3,4`

- Likewise, >_NEXT(LOG) will reject this rule for all of the BPMs for the same reasons described above.

10: `if !duration > 1000 then B-C;  // A:4; R:1,2,3`

- BPMs 2, of course >_NEXT(LOG) will accept this rule for BPM 2. Remember, this rule means "if at any point during this instance, if the running duration value is > 15 then `B-C`". Therefore, if the rule's condition evaluates to true *after* B or C has happened, then this rule is ignored and is not considered to be applied. Likewise, if the condition evaluates to true *before* B or C, then the instance's BPM adapts into BPM 9.
- Likewise, >_NEXT(LOG) will reject this rule for all of the other BPMs for the same reasons described before.

Note: it's assumed that rules don't 'overlap' each other, i.e., if rule x is triggered, it has no impact on whether rule y is triggered.

However, in >_NEXT(LOG), rules are done in sequential order and they *will* be applied with each other. Hence, suppose look at the following for BPM 2;

```
1 if A#duration > 515 then insert F (#duration ?N(400 50));
2 // ^^ Applied like normal
3 if B@duration > 1000 then skip D;
4 // ^^ Is much more likely if the first rule was triggered.
```

Again, this was assumed to not happen, but nonetheless I think this is desirable (thus allowing the user to create rules that may trigger other rules).

2.7 Limitations / Assertions

Of course, the software tool >_next(log) possesses several limitations and assertions that should be taken into consideration before using it;

1. **Non-overlapping rules assumption**: As stated above, >_next(log) assumes that the rules defined by the user do not overlap. If there are overlapping rules, unpredictable or ambiguous outcomes may occur, potentially affecting the accuracy and reliability of the generated logs.
2. **Limited compatibility with input files**: While >_next(log) is designed to work with any input .bpmn (Business Process Model and Notation) files, it has been primarily created and tested using files generated with the bpmn.io tool. Similarly, the software should function with any .mxml files (eXtensible Markup Language for Mining), but it has been predominantly tested with files produced by the BIMP tool. It is essential to note that when using other BPMN or generative MXML tools, there may be minor issues - however, if problems do arise, they *should* be quite easy to fix in the code.
3. **Assumption about Start and End events**: >_next(log) assumes the standard output format of Start and End nodes/events generated by bpmn.io. Consequently, the software references these events as `_Start` and `_End`. It is important to adhere to this naming convention.

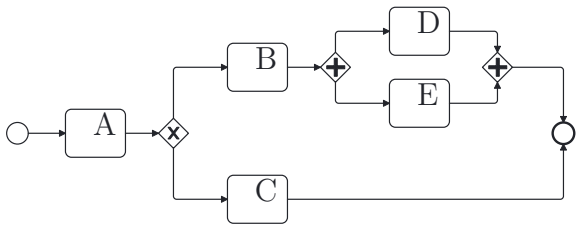
   Therefore, please note: when using bpmn.io, do **not** change the Start/End events default names!

   Likewise, do not name any events "`_Start`" or "`_End`".
4. **Absence of looping in BPMs**: The tool assumes that there are no loops within the Business Process Models (BPMs). If the BPM contains loops, >_next(log) may not generate accurate adapted logs or encounter difficulties in processing the model.
5. **Recognition of specific gateways**: >_next(log) only recognises parallel and exclusive (XOR) gateways. It does not support other types of gateways, which may limit the tool's applicability to BPMs that utilise different gateway types.
6. **No gateway nesting assumption**: >_next(log) assumes that there is no nesting of gateways within the BPMs. This includes the

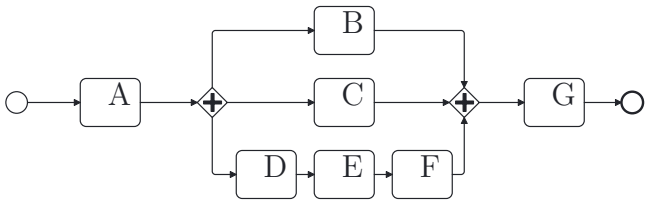
presence of series events nested within gateways.

Incorrectly structured BPMs with gateway nesting, as illustrated in BPMs 12 and 13, may result in unexpected behaviour or errors.

BPM 12



BPM 13



While the software may still function in some cases, its performance and accuracy cannot be guaranteed when encountering nested gateways as its not been tested.

7. **Hardcoded attributes**: >_next(log) includes two hardcoded attributes, namely `#occurred` and `#duration`. Consequently, if these attributes are already defined in the .mxml logs, conflicts may arise. It is important to avoid using these attributes in the .mxml files to prevent any issues during log generation.

8. **Dependency on specific attributes for duration calculation**: The `#duration` attribute in >_next(log) is determined by calculating the difference between the `#start_time` and `#end_time` of each instance defined in the uploaded .mxml logs. Therefore, it is essential to ensure the presence of these two attributes in every .mxml log for accurate duration calculations.

9. **Event definitions**: As seen in the grammar defined earlier, the naming of events within the BPM or in the uploaded .mxml files must adhere to specific naming conventions:
    - Event names should consist of only letters, underscores, and numbers.
    - Event names must start with a letter or an underscore.
    - Hence, event names cannot include spaces, brackets, or any special characters.

By considering the above limitations, users of >_next(log) can effectively assess its applicability, and avoid potential issues when generating logs for adaptive business processes.

3 Evaluation

In this section, we present the evaluation of >_next(log) by showcasing examples of logs that have been adapted using the tool's functionalities. These examples will demonstrate the effectiveness of >_next(log) in modifying event logs based on user-defined rules.

To provide a comprehensive overview, we will highlight an example for each type of possible action available in >_next(log) (ie; `skip`, `insert`, `+ // to parallel`, `- // to series`.

Please note that due to the nature of the adapted logs (and their output), it can be challenging to present them visually on paper or in a limited space. Therefore, we have selected relatively simple examples that effectively illustrate the adaptation process and the resulting changes in the event logs.

Likewise, we have selected a subset of interesting traces that specifically highlight the adaptions performed. These examples serve as representative cases that demonstrate the capabilities of >_next(log).

As mentioned earlier, all initial logs were generated using BIMP. Where applicable, we will highlight specific parameters that were used during the creation of these initial logs using BIMP. Moreover, for simplicity, events will only have the attributes `duration`, `start_time`, `end_time`, `occurred` and hence, we will only create rules based on `duration`.

3.1 Examples of Adapted Logs

Note: All `duration`s are in seconds.

3.1.1 `skip`

BPM 14: `X#duration` $\sim N(500, 50)$ $\forall$ X $\in$ {A,B}.

Rule:

```
if A#duration > 575 then skip B;
```

Original:

trace_id	A#duration	B#duration	_End#start_time	_End@duration	_Path
60	519.159	420.243	2023-07-18T10:55:39.402+00:00	939.402	_Start,A,B,_End
61	471.587	493.18	2023-07-18T10:57:44.767+00:00	964.767	_Start,A,B,_End
72	508.633	523.475	2023-07-18T11:17:12.108+00:00	1032.108	_Start,A,B,_End
30	605.25	476.711	2023-07-18T10:08:01.961+00:00	1081.961	_Start,A,B,_End
51	525.923	454.478	2023-07-18T10:41:20.401+00:00	980.401	_Start,A,B,_End
3	577.222	485.867	2023-07-18T09:22:43.089+00:00	1063.089	_Start,A,B,_End

Adapted:

trace_id	A#duration	B#duration	_End#start_time	_End@duration	_Path	_Label
60	519.159	420.243	2023-07-18T10:55:39.402+00:00	939.402	_Start,A,B,_End	0
61	471.587	493.18	2023-07-18T10:57:44.767+00:00	964.767	_Start,A,B,_End	0
72	508.633	523.475	2023-07-18T11:17:12.108+00:00	1032.108	_Start,A,B,_End	0
30	605.25	0	2023-07-18T10:00:05.250000+00:00	605.25	_Start,A,_End	1
51	525.923	454.478	2023-07-18T10:41:20.401+00:00	980.401	_Start,A,B,_End	0
3	577.222	0	2023-07-18T09:14:37.222000+00:00	577.222	_Start,A,_End	1

Please remember that the tables provided above have undergone filtering, resulting in the exclusion of numerous traces and columns. This selection aims to try highlight elements for demonstration purposes.

As we can see, traces 30 and 3 trigger the given rule.

The "_Path" column correctly updates, as does the "_Label" column.

When B was skipped, its `duration` becomes 0 (as does its `start_time` and `end_time`).

Finally, note how `_End#start_time` and `_End@duration` gets updated accordingly.

3.1.2 `insert`

BPM 15: `A#duration` $\sim N(500, 50)$.

Rule:

```
if A#duration > 575 then insert B (#duration ?N(100 10));
```

Original:

trace_id	A#duration	B#duration	_End#start_time	_End@duration	_Path
60	357.334	0	2023-07-18T10:45:57.334+00:00	357.334	_Start,A,_End
61	589.791	0	2023-07-18T10:51:29.791+00:00	589.791	_Start,A,_End
62	607.015	0	2023-07-18T10:53:27.015+00:00	607.015	_Start,A,_End
25	539.424	0	2023-07-18T09:50:39.424+00:00	539.424	_Start,A,_End
26	513.635	0	2023-07-18T09:51:53.635+00:00	513.635	_Start,A,_End

Adapted:

trace_id	A#duration	B#duration	_End#start_time	_End@duration	_Path	_Label
60	357.334	0	2023-07-18T10:45:57.334+00:00	357.334	_Start,A,_End	0
61	589.791	95.2714744	2023-07-18T10:53:05.062474+00:00	685.0624744	_Start,A,B,_End	1
62	607.015	109.3324325	2023-07-18T10:55:16.347433+00:00	716.3474325	_Start,A,B,_End	1
25	539.424	0	2023-07-18T09:50:39.424+00:00	539.424	_Start,A,_End	0
26	513.635	0	2023-07-18T09:51:53.635+00:00	513.635	_Start,A,_End	0

Please remember that the tables provided above have undergone filtering, resulting in the exclusion of numerous traces and columns. This selection aims to try highlight elements for demonstration purposes.

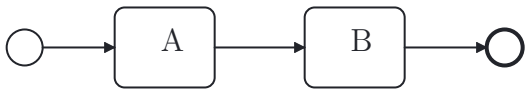
We see traces 61 and 62 trigger the given rule.

The "_Path" column correctly updates, as does the "_Label" column.

Likewise, notice that when B was inserted, its `duration` takes a random value x such that $x \sim N(100, 10)$.

Finally, note how `_End#start_time` and `_End@duration` gets updated accordingly.

3.1.3 `+ // to parallel`



BPM 16: `A#duration` $\sim N(500, 50)$; `B#duration` $\sim N(600, 10)$

Rule:

```
if A#duration > 575 then insert A+B;
```

Original:

trace_id	A#duration	B#duration	B#start_time	_End#start_time	_End@duration	_Path
51	468.458	604.943	2023-07-18T10:32:48.458+00:00	2023-07-18T10:42:53.401+00:00	1073.401	_Start,A,B,_End
90	584.871	595.692	2023-07-18T11:39:44.871+00:00	2023-07-18T11:49:40.563+00:00	1180.563	_Start,A,B,_End
94	518.986	609.704	2023-07-18T11:45:18.986+00:00	2023-07-18T11:55:28.690+00:00	1128.69	_Start,A,B,_End
41	471.941	599.238	2023-07-18T10:16:11.941+00:00	2023-07-18T10:26:11.179+00:00	1071.179	_Start,A,B,_End
80	594.109	592.985	2023-07-18T11:23:14.109+00:00	2023-07-18T11:33:07.094+00:00	1187.094	_Start,A,B,_End

Adapted:

trace_id	A#duration	B#duration	B#start_time	_End#start_time	_End@duration	_Path	_Label
51	468.458	604.943	2023-07-18T10:32:48.458+00:00	2023-07-18T10:42:53.401+00:00	1073.401	_Start,A,B,_End	0
90	584.871	595.692	2023-07-18T11:30:00+00:00	2023-07-18T11:39:55.692000+00:00	1180.563	_Start,A,B,_End	1
94	518.986	609.704	2023-07-18T11:45:18.986+00:00	2023-07-18T11:55:28.690+00:00	1128.69	_Start,A,B,_End	0
41	471.941	599.238	2023-07-18T10:16:11.941+00:00	2023-07-18T10:26:11.179+00:00	1071.179	_Start,A,B,_End	0
80	594.109	592.985	2023-07-18T11:13:20+00:00	2023-07-18T11:23:14.109000+00:00	1187.094	_Start,A,B,_End	1

We see traces 90 and 80 trigger the given rule.

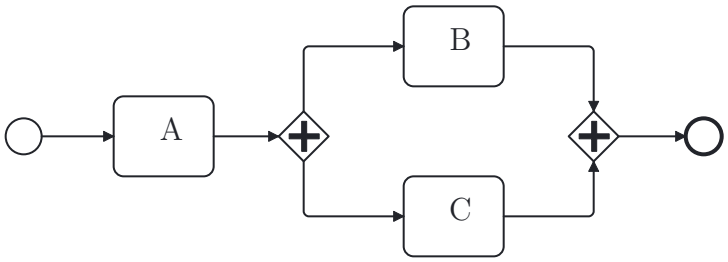
The "_Path" column correctly updates, as does the "_Label" column.

Likewise, notice that when A and B were made parallel, A obviously kept the same `start_time` (unseen above, but can view in the actual adapted logs), however B was shifted accordingly.

Note how something interesting happened in the implementation between trace 90 and 80. Although it may be difficult to observe, in trace 90, `B#end_time` occurs *later* than `A#end_time`. Consequently, in this specific case, `_End#start_time` == (`B#end_time` + `B#duration`). On the other hand, in trace 80, `A#end_time` happens *later* than `B#end_time`. Therefore, in this particular instance, `_End#start_time` == (`A#end_time` + `A#duration`). This would be consistent with however many events you turned into parallel, and is of course unique to each individual trace. Waiting time would also be preserved (there is none in this example).

Please also note that `_End@duration` remains *unchanged* - this behaviour is not a bug! Rather, it is a deliberate design choice. Remember that the `@` symbol represents the accumulative token, indicating the total amount of an attribute used up to that point. Therefore, even though the adapted processes may finish earlier, the total `duration` remains the same. This functionality is intentional since, for example, if events A and B were parallelised, the `_End@cost` (or any other attribute) should rightly remain unchanged, as that was the amount of that attribute used.

3.1.4 `- // to series`



BPM 17: `X#duration` $\sim N(500, 50)$ $\forall$ X $\in$ {A,B,C}.

Rule:

```
if A#duration > 575 then C-B;
```

Original:

trace_id	A#duration	B#start_time	C#start_time	_End#start_time	_Path
71	520.642	2023-07-18T11:07:00.642	2023-07-18T11:07:00.642	2023-07-18T11:15:58.048	_Start,A,B,C,_End
36	586.43	2023-07-18T10:09:46.430	2023-07-18T10:09:46.430	2023-07-18T10:17:58.612	_Start,A,B,C,_End
37	519.05	2023-07-18T10:10:19.050	2023-07-18T10:10:19.050	2023-07-18T10:18:08.616	_Start,A,B,C,_End
99	578.405	2023-07-18T11:54:38.405	2023-07-18T11:54:38.405	2023-07-18T12:03:59.332	_Start,A,B,C,_End

Adapted:

trace_id	A#duration	B#start_time	C#start_time	_End#start_time	_End@duration	_Path	_Label
71	520.642	2023-07-18T11:07:00.642	2023-07-18T11:07:00.642	2023-07-18T11:15:58.048	1556.303	_Start,A,B,C,_End	0
36	586.43	2023-07-18T10:17:58.612	2023-07-18T10:09:46.430	2023-07-18T10:26:10.794	1560.992	_Start,A,C,B,_End	1
37	519.05	2023-07-18T10:10:19.050	2023-07-18T10:10:19.050	2023-07-18T10:18:08.616	1451.61	_Start,A,B,C,_End	0
99	578.405	2023-07-18T12:02:39.537	2023-07-18T11:54:38.405	2023-07-18T12:12:00.464	1620.464	_Start,A,C,B,_End	1

In the given output, we observe that traces 36 and 99 trigger the specified rule. The "_Path" column correctly updates, following the order specified in the rule (e.g., `.. C-B`).

Moreover, the "_Label" column appropriately reflects the updates made according to the rule.

Additionally, it is worth noting that when event B is placed in series (after) event C, event C retains the same `start_time`, while event B (and the "_End" event) are shifted accordingly. This would be consistent with however many events you turned into series and is unique to each individual trace. Furthermore, any waiting time is preserved (in this example, there is none).

As mentioned earlier, it is important to reiterate that the `_End@duration` remains *unchanged* despite any adaptations made.

To emphasize, modifying the rule to `if A#duration > 575 then B-C;` would yield different results.

3.2 Integration with ML.LOG

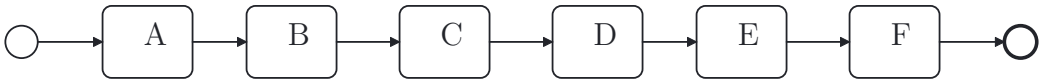
In this section, we discuss the integration of >_next(log) with the ML.LOG software. While ML.LOG is not the focus of my thesis, it is relevant as it serves as the likely subsequent tool to >_next(log).

The integration involves utilising the adapted logs generated by >_next(log) as input for log analysis in ML.LOG. Using machine learning techniques, ML.LOG aims to identify and extract patterns / rule injections from the logs. However, for a more comprehensive analysis and detailed insights, I encourage you to refer to Radu Andrei Sterie's paper [3], where ML.LOG's results are extensively discussed.

3.2.1 Dataset 1

Radu Andrei Sterie and I generated three datasets to assess the performance of ML.LOG in conjunction with >_next(log). However, for the purpose of this thesis, my analysis will concentrate solely on dataset 1, offering a succinct overview of the integration between >_next(log) and ML.LOG as well as the resulting insights.

Below describes how dataset 1 was generated (and then adapted).



BPM 18: Dataset 1's BPM; where `X#duration` $\sim N(500, 50)\ \forall$ X $\in$ {A,..,F}.

Rule:

```
if C#duration > 575 then insert K (#duration ?N(100 15));
```

Original:

trace_id	C#duration	D#start_time	K#duration	_End#start_time	_End@duration	_Path
184	520.272	2023-07-19T22:02:36.535	0	2023-07-22T22:51:24.642	2877.945	"_Start,A,B,C,D,E,F,_End"
27	576.532	2023-07-18T08:37:50.625	0	2023-07-19T17:25:32.212	2981.662	"_Start,A,B,C,D,E,F,_End"
30	478.811	2023-07-18T09:52:33.449	0	2023-07-19T19:59:50.737	3013.945	"_Start,A,B,C,D,E,F,_End"
167	603.946	2023-07-19T18:43:35.746	0	2023-07-22T20:14:34.943	3018.123	"_Start,A,B,C,D,E,F,_End"

Adapted:

trace_id	C#duration	D#start_time	K#duration	_End#start_time	_End@duration	_Path	_Label
184	520.272	2023-07-19T22:02:36.535	0	2023-07-22T22:51:24.642	2877.945	"_Start,A,B,C,D,E,F,_End"	0
27	576.532	2023-07-18T08:39:32.704908	102.079	2023-07-19T17:27:14.291908	3083.741908	"_Start,A,B,C,K,D,E,F,_End"	1
30	478.811	2023-07-18T09:52:33.449	0	2023-07-19T19:59:50.737	3013.945	"_Start,A,B,C,D,E,F,_End"	0
167	603.946	2023-07-19T18:45:20.822520	105.076	2023-07-22T20:16:20.019520	3123.19952	"_Start,A,B,C,K,D,E,F,_End"	1

ml.log took the adapted logs (without knowing the adaption rule) and then utilised four different machine learning techniques, namely Decision Trees, Random Forest Classifiers, KNN, and GLVQ. Each technique was tested with various parameters to explore their performance. However, as stated before, I won't go into the parameters and techniques used [3].

3.2.2 Results

Below, I will provide a summary of the results obtained using dataset 1.

This summary will highlight the performance of the four machine learning techniques provided.

Then we will see if ml.log was able to "extract" the adaption rule.

Decision Trees:

	precision	recall	f1-score	support
class 0	1.00	0.99	0.99	276
class 1	0.92	0.96	0.94	24
macro avg	0.96	0.98	0.97	300
weighted avg	0.99	0.99	0.99	300

```
Train Accuracy: 1.000; Test Accuracy: 0.987
```

Random Forest Classifier:

	precision	recall	f1-score	support
class 0	1.00	1.00	1.00	276
class 1	1.00	1.00	1.00	24
macro avg	1.00	1.00	1.00	300
weighted avg	1.00	1.00	1.00	300

```
Train Accuracy: 1.000; Test Accuracy: 1.000
```

KNN:

	precision	recall	f1-score	support
0.0	0.99	1.00	0.99	276
1.0	1.00	0.83	0.91	24
macro avg	0.99	0.92	0.95	300
weighted avg	0.99	0.99	0.99	300

```
Train Accuracy: 1.000; Test Accuracy: 0.987
```

GLVQ:

	precision	recall	f1-score	support
class 0	0.95	1.00	0.97	276
class 1	1.00	0.33	0.50	24
macro avg	0.97	0.67	0.74	300
weighted avg	0.95	0.95	0.93	300

```
Train Accuracy: 0.976; Test Accuracy: 0.947
```

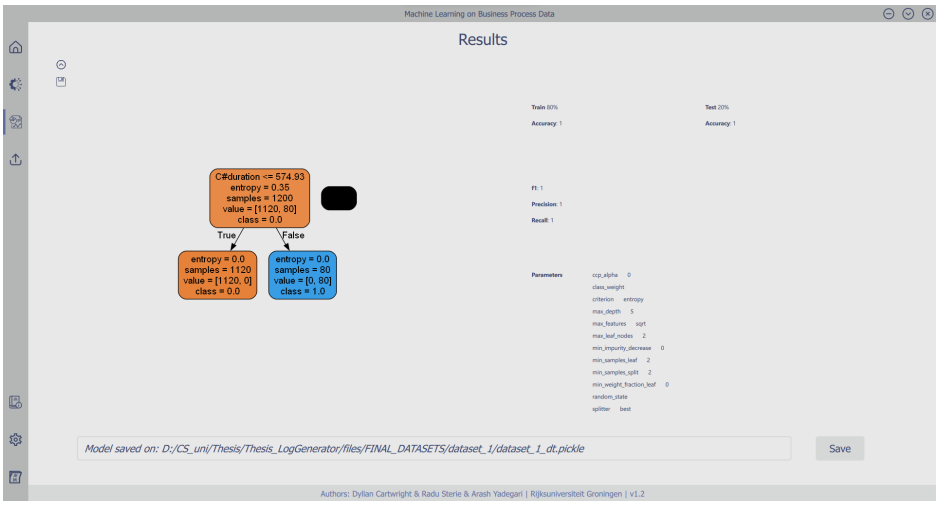


Figure 19: A Found Decision Tree within ML.LOG; $max\_leaf\_nodes = 2$.

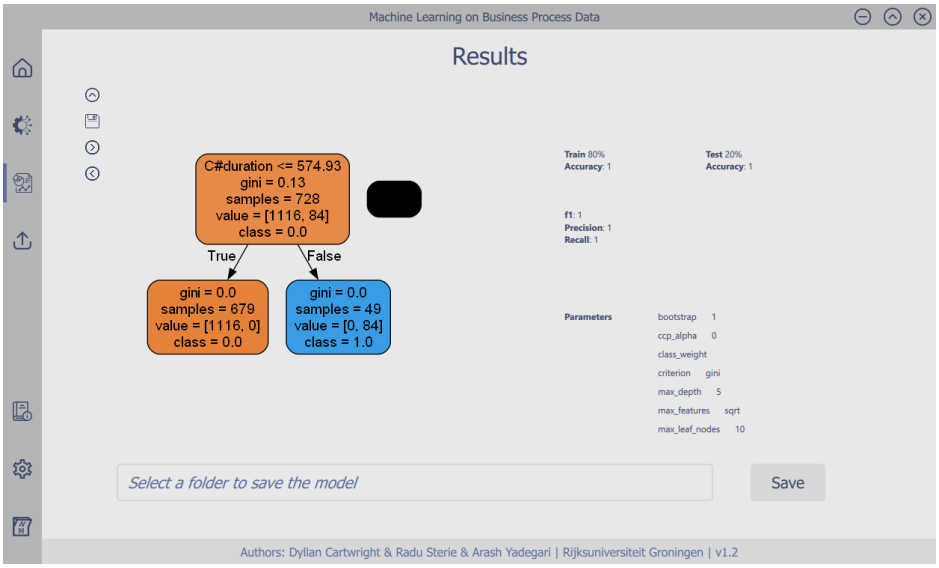


Figure 20: One of the Found Trees using RFC within ML.LOG.

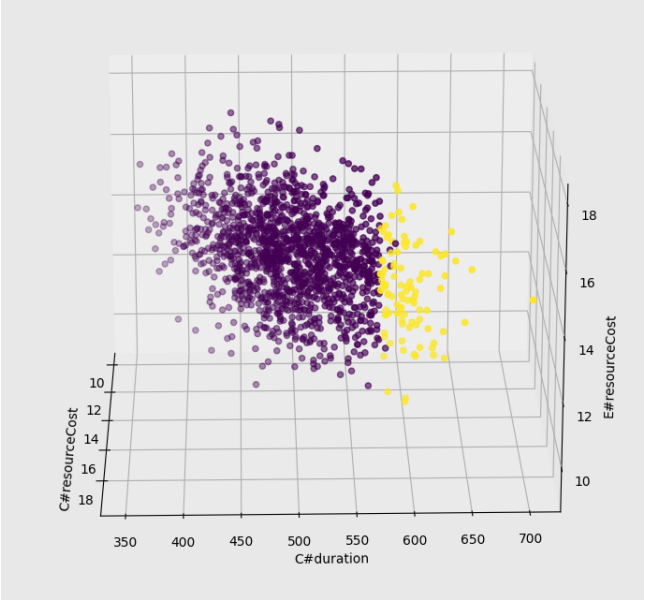


Figure 21: Visualisation of the Labels assigned by KNN within ml.log.

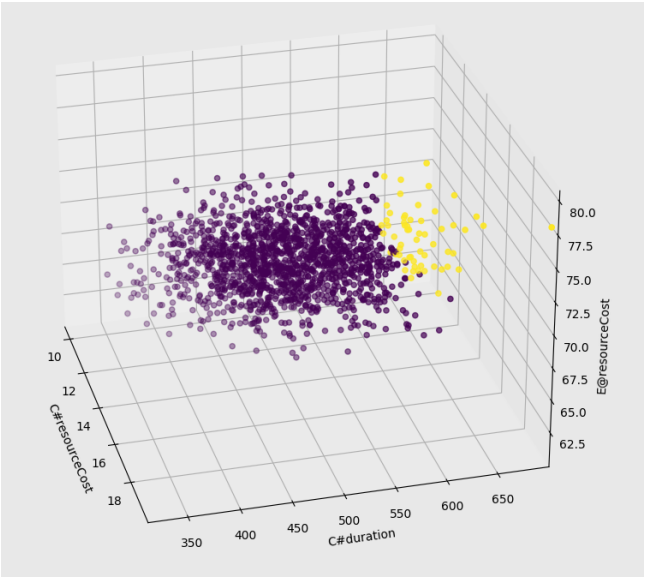


Figure 22: Visualisation of the Labels assigned by GLVQ within ml.log.

Rule Extraction

Hence, upon examining figure 19, it is evident that the "extracted" rule is indeed correct, aligning with the expected behavior.

Similarly, figure 20 showcases another accurate extracted rule. However, it is worth noting that additional trees with more nodes were also discovered in the process. These trees also had perfect accuracy, however, the extra nodes lacked significance and instead represent random correlations that were identified.

It is difficult to properly extrapolate rules from KNN, however, ml.log offers the ability to visualise the labels assigned by KNN. By examining figure 21, we observe that KNN identifies the importance of `C#duration` , seemingly showcasing an invisible - but distinct - boundary at ≈ 575. Consequently, this enables us to "extract" the presence of a rule associated with this KPI and threshold value pairing.

Similar to KNN, GLVQ exhibits a similar pattern (figure 22).

Although this was a very trivial setup / dataset, it is worth noting that all techniques demonstrated very high accuracy.

4 Conclusion

In this concluding section, we summarise the key findings, >_next(log)'s success, and possible future work.

4.1 Revisiting Objectives and Evaluations

The objective of this thesis was to provide users with the flexibility to generate initial business process logs using any preferred method and subsequently adapt these logs based on their own customisable rules, whilst ensuring an intuitive and coherent user interface.

As demonstrated in the examples of adapted logs presented in section 3.1, all adapted logs performed as desired, successfully implementing the defined actions (`insert`, `skip`, `-`, `+`).

As seen in section 3.2 the logs generated for the testing of ml.log also functioned as expected.

Additionally, throughout the development process, logs created within the specified limitations outlined in section 2.7 delivered the intended results. This confirms that >_next(log) successfully meets its goal of enabling users to adapt logs. Furthermore, the user-friendly and streamlined UI of >_next(log) probably surpasses expectations, offering an aesthetic and enjoyable pipeline. The "Rules Editor" page, supported by a strict grammar, detects and highlights rule errors fairly well.

Overall, I'd suggest the outcome of this project can be regarded as a success.

4.2 Future Work

Before exploring future work with regards to >_next(log), it is important to acknowledge that the current capabilities of >_next(log) are limited to very simple and trivial adaptation options. Hence, it should be obvious that the tool currently lacks the ability to create complex / realistic adaptations.

However, >_next(log) serves as a proof of concept, demonstrating the feasibility of developing a tool that enables dynamic log adaptation with user-defined rules. This should stand as motivation to look into extending it to handle more sophisticated and realistic adaptation scenarios.

Ultimately, >_next(log) - if expanded upon - would provide a valuable platform for developers and researchers to accurately test and build tools like ml.log, where creating a tool like ml.log is the *true* goal. As such tools have the potential to revolutionise the field of Business Process Management.

Generating Many Adaptions

Of course, >_next(log) was designed to allow users to not need to code, and just use the UI to adapt logs. However, the UI and backend have been designed to be modular, allowing for separate interactions with the backend through its API. This flexibility opens up possibilities for future work where users can develop their own scripts to interact with the backend.

One potential application of this approach is the generation of a large number of slightly different rules by creating all possible combinations. By utilising a custom script that interacts with the backend of >_next(log), users can automate the process of generating adaptations for each rule. This would

enable the generation of thousands of adapted logs in an incredibly efficient manner, enhancing the scalability and speed of the entire pipeline.

This approach offers significant potential for further exploration and optimisation in terms of rule generation and adaptation. It provides a way to experiment with different rule configurations and evaluate the impact of various adaptations on business process logs.

Below shows a simple (pseudocode) implementation of this:

```
def main():
    vals = [20, 30, 40]
    bpmn_file_path = "path_to_bpmn"
    log_file_path = "path_to_log"
    # Instantiate LogApi
    log_api = LogApi(bpmn_file_path, log_file_path, ...)
    for v in vals:
        generate_all_adaptions(log_api)
        # Could make the calls parallesised / threaded if speed an issue

def generate_all_adaptions(log_api, bpmn_file_path, log_path, v):
    # Generate rules by looping through events, actions, etc
    attributes, event_names = log_api.get_attributes_and_event_names()
    rules = []
    for event_a in event_names:
        for event_b in event_names:
            for attr in attributes:
                for action in ['skip', 'insert', ...]
                    if action == 'skip':
                        rule = generate_skip_rule(event_a, event_b, attr, v)
                    elif ...:
                        ...
                    ...
                    resp = log_api.parse_rules(rule)
                    if resp == 'Accepted':
                        rules.append(rule)

    for rule in rules:
        output_file_path = f"adapted_logs_{rule}.csv"
        log_api.generate_log_files(output_file_path, rule)

def generate_skip_rule(event_a, event_b, attr, v):
    return f"if {event_a}#{attr} > {v} then skip {event_b};"

# Can also add for-loops for different operator types, or
# different token types (#,@,!) etc
```

4.2.1 Improvements

- **Process Mining:**

  Adding a process discovery component to >_next(log) would be a valuable enhancement as it would eliminate the requirement for a separate BPMN process model input, thereby improving the tool's usability and flexibility. By leveraging existing process discovery algorithms like PM4PY, >_next(log) could autonomously infer the underlying process model directly from the given process event logs. Integrating process mining capabilities into >_next(log) could be relatively straightforward since the tool already works with process event logs and is well-structured to accommodate modular development.

- **Support for different input/output formats:**

  Currently, >_next(log) only supports MXML as input, and its output is only CSV. A future improvement could be to enhance flexibility by supporting various input/output formats, such as XES, CSV, or other commonly used formats in the Business Process Management field, see figure 23.

- **Extension of rule grammar and actions:**

  The rule grammar in >_next(log) could be extended to include more actions beyond the existing ones. This would allow users to define a wider range of rule-based adaptations, providing more flexibility and customization options.

- **Enhancement of accepted gateway types:**

  Currently, >_next(log) only supports XOR and parallel gateways. An improvement could be to expand the accepted gateway types to include more advanced types, such as event-based gateways.

- **Support for more complicated process structures:**

  >_next(log) could be enhanced to handle more complicated process structures, such as allowing nesting or looping in the processes. This would enable the adaptation of processes with more intricate control flow patterns, where sub-processes can be defined within main processes or certain activities can be repeated based on specific conditions. This

would be one of the hardest improvements, but would make >_next(log) a lot more realistic.

Finally, more testing, bug fixing, and the addition of (more) error handling would contribute to the improvement of >_next(log).

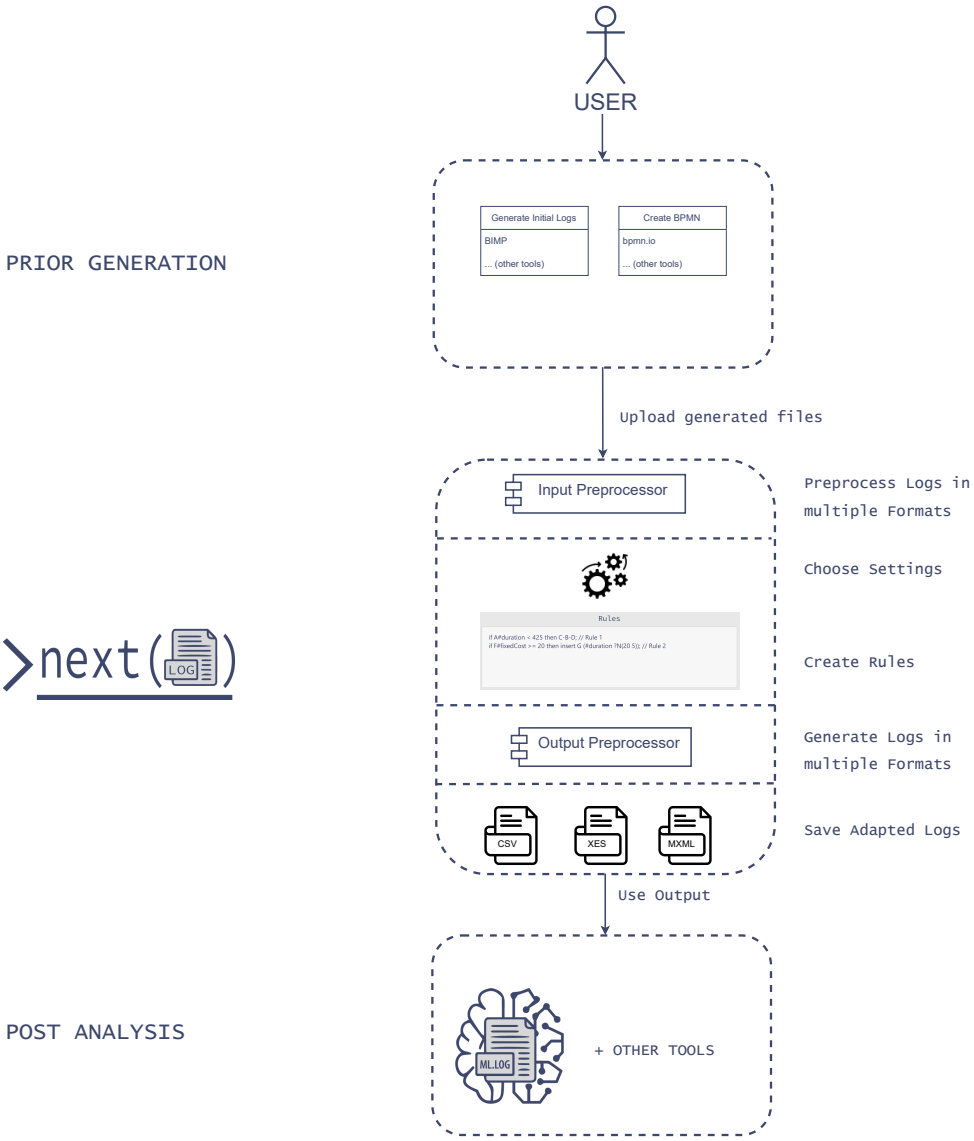


Figure 23: The (Ideal) Flow/Pipeline of >_next(log)

4.3 Auxiliary Information

Division of Tasks

I was the only student who was formally responsible for this project, so there were no division of tasks. Although, do note, Radu and I shared code/ideas throughout, and eventually combined >_next(log) and ml.log into one program.

Deliverables

- The final thesis.
- Software source code (and/or a compiled program).
- Datasets generated.

All will be emailed to Arash Yadegari, nonetheless, can also be found here.

Grading

- Scientific quality of research and technical contribution: 40%.
- Project management and interpersonal skills: 20%.
- Final presentation: 20%.
- Report/Thesis: 20%.

Ethical Declaration

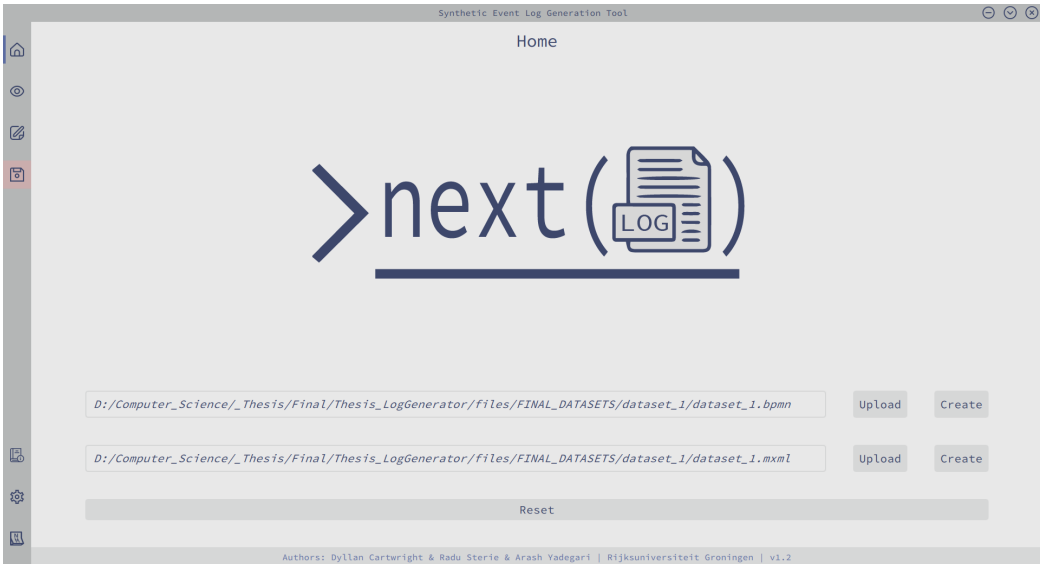
I hereby declare that this thesis presented herein is the result of my original research work. I assert that this thesis has not been submitted, either wholly or partially, for any other academic degree or qualification at any other university or institution. All sources, including published or unpublished works, have been duly acknowledged and referenced, except where explicitly indicated.

Dyllan Cartwright, s3479528, Sunday 30th July, 2023:

```
if you#reading == here then TY!
```

A Screenshots of >_next(log)

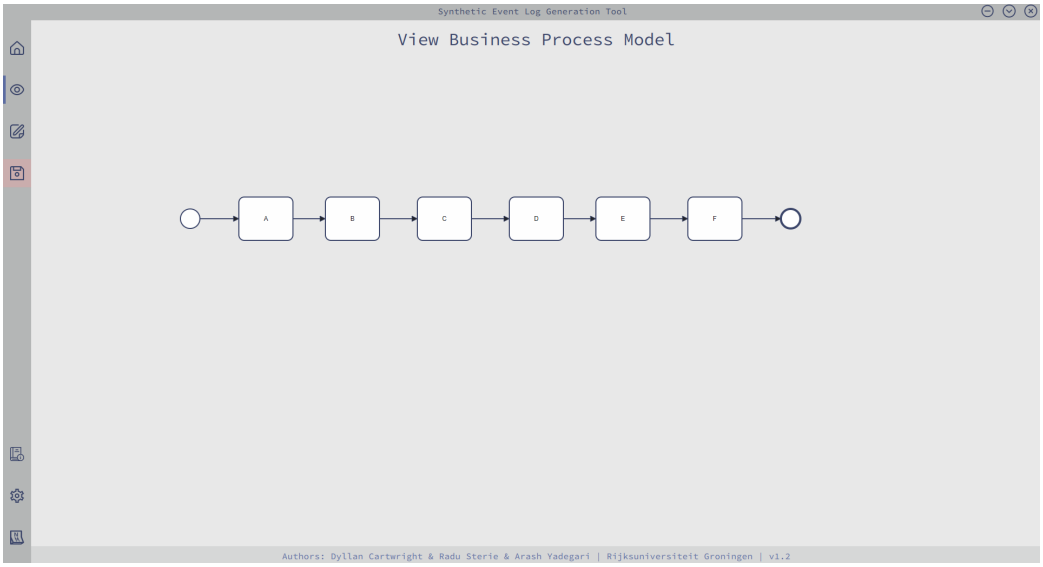
The Home Page



On this page, users can easily upload their .bpmn and .mxml files.

This page serves as the starting point for the log generation process, allowing users to select the necessary files for adaptation.

The View Page



This page provides users with a visual representation of their uploaded BPM.

It enables users to inspect and review the structure and flow of the process model, aiding in the understanding of the process when creating rules.

The Rule Editor Page

On this page, users can create their rules for log adaptation.

The intuitive interface allows users to define rules based on specific conditions and actions.

After creating the rules, users can click ‘Parse Rules’ to verify their validity.

By clicking ‘Available’, users can view all the defined events and attributes to facilitate rule creation.

The Save Page

This page is where users can generate and save the adapted logs.

Users can specify the desired output file name and save the adapted logs as a .csv file.

Additionally, users have the option to save the "original" unadapted logs as a separate .csv file, providing traceability and allowing for debugging purposes if needed.

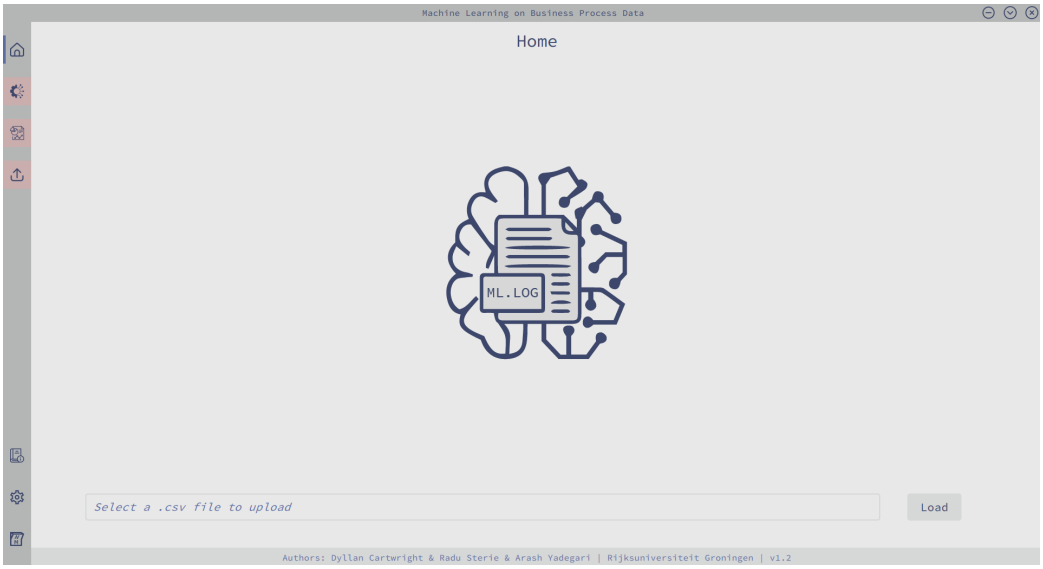
The Settings Page

This page offers users the ability to customise certain preferences according to their requirements.

Users can define the default folder location for file uploads.

If the default folder location setting is disabled, >_next(log) will remember the last folder used for uploading files and will open there.

Switching to ml.log



The Switch Icon allows users to toggle between >_next(log) and ml.log. The above is ml.log's home page.

References

[1] A. Y. Ghahderijani and D. Karastoyanova, "Synthetic event log generation for kpi-based process adaptations using simulation," *University of Groningen*, 2023. *Unpublished as of yet.

[2] A. Y. Ghahderijani and D. Karastoyanova, "Applying decision trees as a mean for correlation identification and learning from adapted business process cases," *University of Groningen*, 2023. *Unpublished as of yet.

[3] R. A. Sterie, "Adaptive business process analysis using machine learning algorithms," *University of Groningen*, 2023. *Unpublished as of yet.

[4] M. Dumas, M. L. Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management*. Springer Publishing Company, Incorporated, 2018.

[5] M. Weske, *Business Process Management: Concepts, Languages, Architectures*. Springer Berlin Heidelberg, 2019.

[6] M. Dumas and J. Mendling, *Business Process Event Logs and Visualization*, pp. 398–409. Cham: Springer International Publishing, 2019.

[7] H. A. Malak, "What is business process simulation? why is it important?," Mar 2023.

[8] P. Henderson, *Systems Engineering for Business Process Change: New Directions*. Springer Science & Business Media, 2012.

[9] I. Beerepoot, C. Di Ciccio, and H. A. R. et al., "The biggest business process management problems to solve before we die," *Computers in Industry*, vol. 146, p. 103837, 2023.

[10] Z. Bozorgi, I. Teinemaa, M. Dumas, M. Rosa, and A. Polyvyanyy, "Process mining meets causal machine learning: Discovering causal rules from event logs," in *ICPM*, p. 28, 2020.

[11] R. Conforti, M. Rosa, and A. T. Hofstede, "Filtering out infrequent behavior from business process event logs," *IEEE Transactions on Knowledge and Data Engineering*, 2017.

[12] M. Röglinger, J. Pöppelbuß, and J. Becker, "Maturity models in business process management," *Business Process Management Journal*, vol. 18, no. 2, pp. 328–346, 2012.

[13] R. Conforti, M. Dumas, L. Garca-Bauelos, and M. La Rosa, "Beyond tasks and gateways: discovering BPMN models with subprocesses, boundary events and activity markers," in *Business Process Management*, vol. 8659 of *Lecture Notes in Computer Science*, pp. 101–117, Springer International Publishing, 2014.

[14] A. K. A. d. Medeiros and C. W. Günther, "Process mining: using CPN tools to create test logs for mining algorithms," in *Proceedings of the Sixth Workshop on the Practical Use of Coloured Petri Nets and CPN Tools (CPN 2005)*, vol. 576 of *DAIMI*, pp. 177–190, University of Aarhus, 2005.

[15] B. P. Zeigler, "Hierarchical, modular discrete-event modelling in an object-oriented environment," *Simulation*, vol. 49, no. 5, pp. 219–230, 1987.

[16] D. Cetinkaya, A. Verbraeck, and M. D. Seck, "Model transformation from BPMN to DEVS in the MDD4MS framework," in *Proceedings of the 2012 Symposium on Theory of Modeling and Simulation - DEVS Integrative M&S Symposium, TMS/DEVS '12*, pp. 28:1–28:6, Society for Computer Simulation International, 2012.

[17] H. Bazoun, Y. Bouanan, G. Zacharewicz, Y. Ducq, and H. Boye, "Business process simulation: transformation of BPMN 2.0 to DEVS models (wip)," in *Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative, DEVS '14*, pp. 20:1–20:7, Society for Computer Simulation International, 2014.

[18] S. Boukelkoul and R. Maamri, "Optimal model transformation of BPMN to DEVS," in *2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA)*, pp. 1–8, 2015.

[19] T. Stocker and R. Accorsi, "SecSy: security-aware synthesis of process event logs," in *Proceedings of the 5th International Workshop on Enterprise Modelling and Information Systems Architectures*, (St. Gallen, Switzerland), 2013.

[20] A. Burattin and A. Sperduti, "PLG: a framework for the generation of business process models and their execution logs," in *BPM 2010 Workshops, Proceedings of the Sixth Workshop on Business Process Intelligence (BPI2010)*, vol. 66 of *Lecture Notes in Business Information Processing*, Springer-Verlag, 2011.

[21] A. Burattin, "PLG2: multiperspective processes randomization and simulation for online and offline settings," tech. rep., CoRR abs/1506.08415, 2015.

[22] A. P. Freitas and J. L. Pereira, "Process simulation support in bpm tools: The case of bpmn," in *Proceedings of 2100 Projects Association Joint Conferences – Vol.X (20XX)*, 2015.