Feature Aggregation in Process Mining
Programmer's Guide to adding new attributes and filters

Inês Cardo, Kai Liehr, Thomas Hsu

January 13, 2022

Contents

1 Introduction **3**

2 Adding new attributes to be augmented **3**
2.1 Programming the derivation method 3
2.2 The necessary UI changes: 3
2.3 Changes if additional input is needed for derivation: 5

3 Adding new filters **5**
3.1 Programming the filter 5
3.2 The necessary UI changes 6
3.3 Changes if additional input is needed for filter 7

1 Introduction

The goal of this document is to guide users of our tool for feature aggregation and clustering of event logs to add new attributes and filters that a log can be augmented or filtered by. This will require some programming experience on the side of the user.

2 Adding new attributes to be augmented

This section explains step by step how to add new attributes that can be derived for event logs.

2.1 Programming the derivation method

1. Choosing an abbreviation:

   All attribute derivations have a corresponding abbreviation consisting of a letter and a number.

   The letter denotes the type of attribute that is derived. There are four possible categories:

   - Time related(abbrv.: T): such as service time, waiting time, etc.
   - Workload related(abbrv.: R): such as the workload of a certain resource or activity
   - Control flow(abbrv.: C): such as previous activity
   - Data attributes(abbrv.: D): such as the max value of a certain attribute in a trace

   The number is just chosen as the next highest free number in that category.

   For the rest of section 2, lets assume the attribute we want to add to be abbreviated *T5.*

2. Writing the method:

   Disclaimer: The names of classes and methods need to be followed, adjusting only for your abbreviation.

   Create a file called *add_T5.py* in the folder *add_Attributes*

   Define a method in that file called $add\_T5(log)$, which receives the log to which it adds attribute $T5$ and then return the log.

   If you need certain info about the log's attributes, global variables are available in the *log_utils.py* file denoting the timestamp, case id, resource, activity and, if there, lifecycle attribute.

   For inspiration on how to work with log objects, useful imports or other issues, take a look at the other files in the folder.

3. Importing the new file:

   Next write the import statement: *from . import add_T5* into the file called *add_Attr.py*.

2.2 The necessary UI changes:

All changes for the UI are to be done in the file *Attributes.html*, which can be found in the html template folder.

1. Add a check dropdown function exactly as seen in the picture 1. You can use copy paste. Then, of course, change to an unused name for example *check_dropdownT5*

2. Change the value in its first line's function(in the picture 1: $'time4 - text'$) to something like $'time5 - text'$.

```
96      function check_dropdown4(){
97          var variavel_text = document.getElementById('time4-text')
98          if (variavel_text.style.display=='none'){variavel_text.style.display='block';}
99          else {variavel_text.style.display='none';}
100     }
```

Figure 1: Check dropdown function example.

```
<div style='position: absolute; top: 27%; left: 65%; right: 5%; text-align: center;'>
    <h1>Attributes to derive:</h1>
    <div style='border: 1px solid black; width: 380px; height: 200px; overflow:auto;'>

        <p id='time1-text' style='display:none'>Activity duration of completed activities(T1)</p>
        <p id='time2-text' style='display:none'>Time since start of case(T2)</p>
        <p id='time3-text' style='display:none'>Time to end of case(T3)</p>
        <p id='time4-text' style='display:none'>Case duration(T4)</p>
        <p id='workload1-text' style='display:none'>Workload of event resource left after event(R1)</p>
        <p id='workload2-text' style='display:none'>Total Workload left after event(R2)</p>
        <p id='data-flow6-text' style='display:none'>Sum value of a certain attribute after current event(D6)</p>
        <p id='data-flow5-text' style='display:none'>Min value of a certain attribute after current event(D5)</p>
        <p id='data-flow4-text' style='display:none'>Max value of a certain attribute after current event(D4)</p>
        <p id='data-flow3-text' style='display:none'>Average value of a certain attribute after current event(D3)</p>
        <p id='data-flow2-text' style='display:none'>Last assigned value of a certain attribute after current event(D2)</p>
        <p id='data-flow1-text' style='display:none'>Last assigned value of a certain attribute prior to event(D1)</p>
        <p id='control-flow1-text' style='display:none'>Number of times a certain activity is executed before an event(C1)</p>
        <p id='control-flow2-text' style='display:none'>Next activity after event(C2)</p>
        <p id='control-flow3-text' style='display:none'>Activity prior to event(C3)</p>
    </div>
    <button type='button' onclick='updateButton()' style='margin-top:10%; width: 380px; border-radius: 8px; padding: 14px 40px; font-
        Update Event Log
    </button>
```

Figure 2: Entries for update button box.

3. Next add a line for your new Attribute derivation function in the update button boxes list as seen in picture 2.

   Naturally with your id and description followed by its abbreviation. As a reminder, in our example the id was $'time5 - text'$ and the abbreviation $T5$.

4. Above the previous change, add an entry into the dropdown menu of the category your attribute is in. For the example it is the time category as seen in picture 3.

   Once again decide on an id for example $time5$ and a description followed by the abbreviation. For the onclick function call the one you wrote, in our example $'check\_dropdownT5()'$.

```
<div style='position: absolute; top: 20%; left: 30%; right: 45%; text-align: center;'>
    <h1>Chose which attributes you want to derive:</h1>
    <div class="sidenav">

        <button class="dropdown-btn">Time
            <i class="fa fa-caret-down"></i>
        </button>
        <div class="dropdown-container">
            <a id='time1' onclick='check_dropdown1()'>Activity duration of completed activities(T1)</a>
            <a id='time2' onclick='check_dropdown2()'>Time since start of case(T2)</a>
            <a id='time3' onclick='check_dropdown3()'>Time to end of case(T3)</a>
            <a id='time4' onclick='check_dropdown4()'>Case duration(T4)</a>
        </div>

        <button class="dropdown-btn" >Workload
```

Figure 3: Entries for dropdown menu.

5. Lastly, add an entry in the function of the update button for your abbreviation to be added

into the list of attributes to be derived. This takes the form of an if condition that checks if the attribute was chosen and, if so, pushes the abbreviation into the list called *listattributes* as seen in picture 4.

In our example the id we need to use is $'time5 - text'$ and the abbreviation is $T5$

```
157     // called upon clicking: Update event log
158     // turns the chosen attributes into javascript list of their abbreviation ID(R1, T3, etc.)
159     // IMPORTANT for functions with additional input, add after their list attribute push, see f
160     // finally does ajax upload
161     function updateButton() {
162
163         const listattributes = []
164         const listExtraInput = []
165
166
167
168
169         if (document.getElementById('time1-text').style.display=='block'){
170             listattributes.push("T1")
171         }
172         if (document.getElementById('time2-text').style.display=='block'){
173             listattributes.push("T2")
174         }
175         if (document.getElementById('time3-text').style.display=='block'){
176             listattributes.push("T3")
177         }
178         if (document.getElementById('time4-text').style.display=='block'){
179             listattributes.push("T4")
180         }
```

Figure 4: Entries for update button function.

2.3 Changes if additional input is needed for derivation:

If you need additional user input for your attribute derivation, there is a way to do it.

1. Add a prompt for user input into the update button's function in *Attribute.html* (same place as last step of prior subsection) as seen in picture 5.

   Basically, add the last three lines into your attribute's if condition and change their description to fit your attribute's abbreviation and description.

2. In *add_Attr.py* in the folder *add_Attributes*:

   Add your abbreviation to the list called *extra_input_needed* as seen in the picture 6.

3. Assuming you adhered to the steps before, all that is left to do is to adjust your method to receive a second argument after the log like for example: *def apply_T5(log, extra_input_string):*.

3 Adding new filters

This section describes step by step how to add new filters.

3.1 Programming the filter

1. Choosing the filter's abbreviation:

   All filters have a corresponding abbreviation starting with the letter F followed by a number unique to that filter. For simplicity just choose the next free increment.

```
        // listExtraInput.push('D1!' + attributeD1)
    }
    if (attributeD2.length!=0){
        listattributes.push("D2")
        console.log("Chosen attribute for D2"+ attributeD2)
        for (let i = 0; i < attributeD2.length; i++) {
            listExtraInput.push('D2!' + attributeD2[i])
        }
    }
    if (attributeD3.length!=0){
        listattributes.push("D3")
```

Figure 5: Prompt for user input in update button function.

```
49      # create actual list from info string via ,
50      extra_info_list = extra_info.split(',')
51
52
53
54      # list of attribute abbreviations that require extra_info
55      extra_input_needed = ['D1','D2','D3','D4','D5','D6']
56
57      # call each chosen function:
58      for abbrv in attr_list:
59
60          name_of_method = "add_" + abbrv
```

Figure 6: List for extra input attributes in *add_Attr.py*.

For the rest of section 3, lets assume the filter we want to add to be abbreviated *F8*.

2. Add the filter method:

   In the file *apply_filters.py* add your method called *apply_F*8 that gets a log-object(pm4py data type) as argument and returns the log object but filtered.

   Pm4py has useful methods such as for filtering traces. Just look at the already existing filters for examples.

3.2 The necessary UI changes

All changes for the UI are to be done in the file *Filters.html*, which can be found in the html template folder.

1. Add a variable and instantiate it with 0 like in the picture 7, for our example:

   var $filter8 = 0$

2. Add a check filter function:

```
<script>

    var filter1 = 0
    var filter2 = 0
    var filter3 = 0
    var filter4 = 0
    var filter5 = 0
    var filter6 = 0
    var filter7 = 0

    function check_filter1(){
        if (filter1==0){
            filter1 = 1
            document.getElementById('filter1').style.color='white'
            document.getElementById('filter1').style.backgroundColor='lightblue'
        }
        else {
```

Figure 7: Variables for the filters in Filters.html.

Like in the picture 8, only change the function name and id. For our example it would be $check\_filter8()$ and $filter8$.

```
46
47      function check_filter2(){
48          if (filter2==0){
49              filter2 = 1
50              document.getElementById('filter2').style.color='white'
51              document.getElementById('filter2').style.backgroundColor='lightblue'
52          }
53          else {
54              filter2 = 0
55              document.getElementById('filter2').style.color='#818181'
56              document.getElementById('filter2').style.backgroundColor='white'
57          }
58      }
```

Figure 8: check filter function in Filters.html.

3. Add an entry to the filter display:

   Add an entry like the others but with in our example $'filter8'$ as the id and the previously created check function being $'check\_filter8()'$ as onclick. Afterwards a short description text with the abbreviation in brackets appended. For reference see picture 9.

4. Add entry in the function of the Filter Event Log button:

   A simple if condition with an expression to push the abbreviation into the list called $listFilters$. Follow picture 10 and adjust to for our example $filter8$ and "$F8$" respectively.

3.3 Changes if additional input is needed for filter

If you need additional user input for your filter, there is a way to do it.

1. Add a choice of attribute value for user input into the filter button's function in $Filter.html$ (same place as last step of prior subsection) as seen in pictures 11,12.

```
{% block content %}


<div style='position: absolute; top: 25%; left: 46%; right: 20%; text-align: center;'>
    <h1>Chose how to filter your event log:</h1>
    <div class="sidenav" style='border: 1px solid black; width: 100%; height:50%; text-align: center'>
        <a id='filter1' onclick='check_filter1()' style='padding:0px 0px 20px 0px; text-align: center;'>Keep All Events(F1)</a>
        <a id='filter2' onclick='check_filter2()' style='padding:0px 0px 20px 0px; text-align: center;'>Only Keep First Event Of A Case(F
        <a id='filter3' onclick='check_filter3()' style='padding:0px 0px 20px 0px; text-align: center;'>Only Keep Last Event Of A Case(F3
        <a id='filter4' onclick='check_filter4()' style='padding:0px 0px 20px 0px; text-align: center;'>Only Keep "Complete" Events(F4)</
        <a id='filter5' onclick='check_filter5()' style='padding:0px 0px 20px 0px; text-align: center;'>Only Keep "Start" Events(F5)</a>
        <a id='filter6' onclick='check_filter6()' style='padding:0px 0px 20px 0px; text-align: center;'>Only Keep Events With A Certain A
        <a id='filter7' onclick='check_filter7()' style='padding:0px 0px 20px 0px; text-align: center;'>Only Keep Events With A Certain F
    </div>

    <button type='button' onclick='filterButton()' style='margin-top:10%; width: 380px; border-radius: 8px; padding: 14px 40px; font-size
        Filter Event Log
    </button>

</div>

<script>
```

Figure 9: Entries for filter display in Filters.html.

2. In *apply_filters.py*:

   Add your abbreviation to the list called *extra_input_needed* as seen in the picture 13.

3. Assuming you adhered to the steps before, all that is left to do is to adjust your method to receive a second argument after the log like for example: *def apply_F8(log, extra_input_string):*.

```
126     // upon clicking filter all previously selected
127     function filterButton() {
128
129         // list all chosen filters
130         const listFilters = []
131         // list for extra input
132         const listExtraInput = []
133
134         if (filter1==1){
135             listFilters.push("F1")
136         }
137
138         if (filter2==1){
139             listFilters.push("F2")
140         }
141
142         if (filter3==1){
143             listFilters.push("F3")
144         }
```

Figure 10: Entries for function of filter button in Filters.html.

```
        listFilters.push("F5")
    }

    if (activityF6!=''){
        listFilters.push("F6")
        console.log("Chosen activity for F6"+ activityF6)
        listExtraInput.push('F6!' + activityF6)
    }

    if (resourceF7!=''){
```

Figure 11: Dealing with user input in filter button function.

```
<button id='filter6' style='padding:0px 0px 20px 0px; text-align: center;' class="dropdown-btn">Only keep events with a certain activity(F6)
    <i class="fa fa-caret-down"></i>
  </button>
  <div id='activitiesF6' class="dropdown-container">

  </div>

<button id='filter7' style='padding:0px 0px 20px 0px; text-align: center;' class="dropdown-btn">Only keep events with a certain resource(F7)
    <i class="fa fa-caret-down"></i>
  </button>
  <div id='resourcesF7' class="dropdown-container">

  </div>
```

Figure 12: Dropdown menu for filters with additional input.

```
    # create actual list from extra_input string via ,
    extra_info_list = extra_input.split(',')

    # list of filter abbreviations that require extra_input
    extra_input_needed = ['F6','F7']


    # call each chosen filter's function:
    for abbrv in filter_list:

        name_of_method = "apply_" + abbrv
```

Figure 13: List for extra input filters in *apply_filters.py*.