CSM: Project Initiation

Odysseas Karanikas
odysseas.karanikas@rwth-aachen.de

Mann, Daniel
daniel.mann@rwth-aachen.de

Daniel Rein
drein99@outlook.de

April 2019

1 Introduction

2 Functional Requirements

The planned software can be understood as a mapping of a given input to an output. In our case, the input is an XES log file and one or more views and the output is a state chart. Since this function is our core feature it is important to discuss the details of input and output.

2.1 Input

The initial input of our software will be an XES file. XES stands for extensible event stream. It is a standard format for the representation of causally linked and ordered events and makes use of the XML format [**xes**]. Generally, the structure of an XES file includes a trace and an event element where an event is a subelement of a trace. An event can contain attributes such as a timestamp, a name and other indivually specified properties. An excerpt of an XES file can look as follows[1]:

.

[1]Usually an XES file contains additional metadata and definition of elements and attributes which we omitted in our example.

```
<trace>
    <string key="concept:name" value="Trace number one"/>
    <event>
        <string key="concept:name" value="Register client"/>
        <string key="system" value="alpha" />
        <date key="time:timestamp"
            value="2009-11-25T14:12:45:000+02:00" />
        <int key="attempt" value="23">
            <boolean key="tried hard" value="false" />
        </int>
    </event>
    <event>
        <string key="concept:name" value="Mail rejection"/>
        <string key="system" value="beta" />
        <date key="time:timestamp"
            value="2009-11-28T11:18:45:000+02:00" />
    </event>
</trace>
```

A trace represents a closed and ordered process in the system while events depict stages or key points in the process. For example in an insurance company, one could consider all milestones of an insurance case as events belonging to the same trace. Another case would then necessitate another trace.

In our case the event log will have a particular semantic. We will understand the events as states that a process goes through where these states are spread in time. That is every event has a duration attribute denoting how long a given process, represented by a trace, spend in the corresponding event. To make this clearer, we will take the example of a grocery store. A trace and thus a process corresponds to a customer pathing through the store. On his way through the store he will go through different locations which will be his states. These locations can be the entrance (S), two different sections of the bakery (B_1 and B_2), two different subsections of the beverage section (G_1, G_2) and finally the cash register (P) and the exit section E. A possible log file can look as follows:

```
   .
Trace    State    Duration
1        S        1
1        B_1      3
1        G_2      4
```

```
1        P        5
1        E        1

2        S        1
2        G_1      3
2        P        4
2        E        1

3        S        1
3        E        1

4        S        1
4        B_2      3
4        P        3
4        E        1
```

Here we chose a different more simple representation that captures the order of states, their respective duration and their affiliated trace.

The log file is not the only input we expect. The user should furthermore be able to create views as discussed in the Use case section. A view is an explicitly non-injective mapping from the initial state space to the target state space. Both the mapping the target state space are determined by the user and comprise the view. We use the example of the grocery store from above and give two example views:

```
   .
States   View1    View2

S        S        S'
B_1      B        S'
B_2      B        S'
G_1      G        G
G_2      G        G
P        E'       E'
E        E'       E'
```

2.2 Output

In the Feasibility Study of the our Project Initiation we already discussed some structure of the output. In general, our output will be a set of graphs depicted in different views. One of these views will be the original view that

is computed from the log file directly. Nodes in this graph represent states in the logs. Each node contains the name of the state, the average duration and the total number of occurrences in the event log. The average is computed over all occurrences. The edges represent state transitions. When in a trace, one state directly follows another one then their corresponding nodes are to be connected by a directed edge where this edge points to the later, successor state. Each such transition is annotated with the relative frequency of transitions to the target state given the total number of occurrences from that state **or the total number of outgoing transitions from that state**. This is illustrated in Figure 1.

The other output graphs correspond to the views as specified in the subsection on the input. In a view, a number of states can be lumped into the same state. In the above example view 1, the states B_1 and B_2 are combined to the state B. The resulting state holds again an average duration time. We imagine two different possibilities for the computation of this value. Either we could average over all occurrences of every state in the combined state or for every trace sum up the total duration time in the combined state and then average over all traces[2]. The total number of occurrences of the composite state is the sum of the number of occurrences of every component state.

We shall investigate the transition frequencies of the composite states further. Let Ω denote the state space. We consider a transition from the composite states $S \subset \Omega$ to $T \subset \Omega$ where we understand both states as sets of basic states. The number of transitions from states all substates of S to all substates of Q is then the number

$$\sum_{(s,q)\in S\times Q} N(s,q)$$

where $N(s,q)$ denotes the number of transitions observed between the original, raw states s and q. The relative frequency of transitions between the composite states S and Q is then given by the above sum divided by the number

$$\sum_{(s,t)\in S\times \Omega} N(s,t)$$

of outgoing transitions from the lumped state S.

[2]We deem the latter option to be more meaningful but leave the discussion to be open until our advisor has been consulted about this issue.

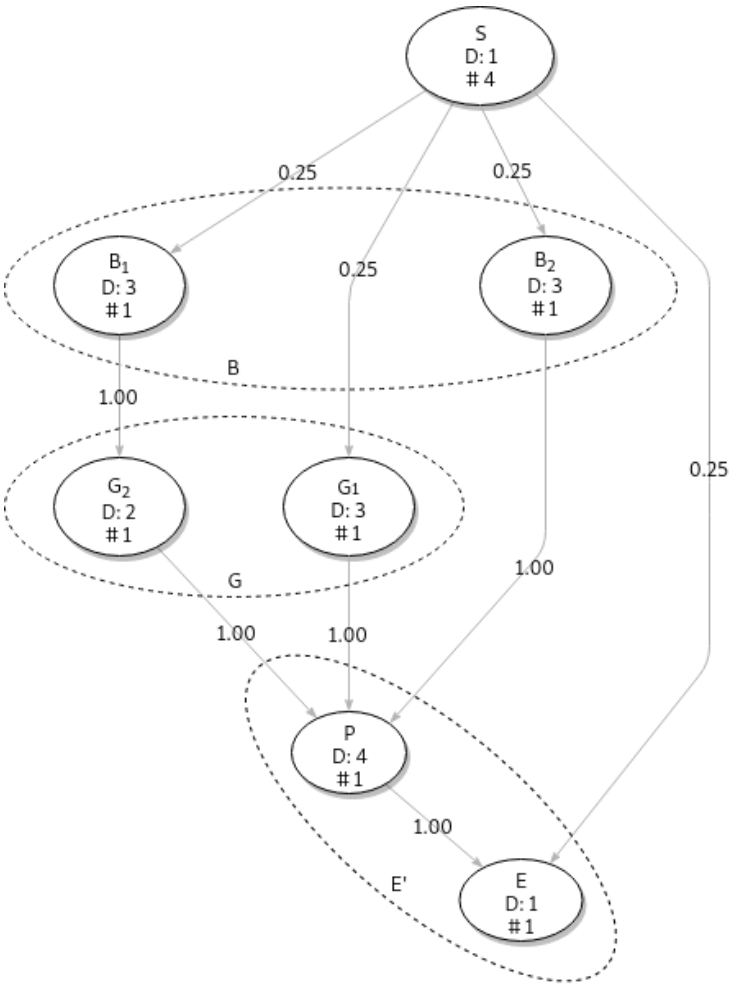

Figure 1: An example output following the log presented above. We see that the transition from S to B_1 is annotated by the numeric value 0.25. This follows from that fact that B_1 followed S in a fourth of the times. The dashed ellipses represent a possible view on could choose.

To illustrate this procedure let us consider the partitioning chosen in figure 1. Here a we have the state space $\Omega = \{S, B_1, B_2, G_1, G_2, P, E\}$[3]. The composite states we chose in the first example view are $B := \{B_1, B_2\}$, $G = \{G_1, G_2\}$ and $E' = \{P, E\}$. The graph is then transformed according to the rules discussed above, yielding the graph in figure

3 Technical Requirement

3.1 Needed software

To deploy the solution there are multiple requirements. Firstly a python version greater 3 is needed. Next there is pip the software that will manage our server application. Now all we need server-wise is django , the project was written under django 2.2 , therefor it is recommended to use at least version 2.2.

3.2 Client-Server architecture

The solution requires a rather extensive client-server architecture for computational and quality of service reasons. Since the main computation is on the server the client would normally need to refresh every time a request is sent. This is not optimal for a mobile work environment since wireless networks are not as consistant as wired ones. Therefor the software only causes the client to do a complete refresh once a major computation needs to be done (e.g. create a new view), but smaller requests (e.g. labeling in current view) are computed client side. Since a project is not stored locally we will implement a reuqest queue that will be used to handle the asynchronous work load. This queue also ensures, that a request is never lost, because an element is not deleted before the corresponding confirmation is received.

3.3 Django management

To connect the front-end with the back-end django will be used as a web server. The main tasks consist out of the following :

- Receive requests and files and sent them to the back-end.

[3]Mind that S is here a basic state while in our notation about S is composite

- Sent processed information to the client in form of a httprequest.