Understanding Spaghetti Models with Sequence Clustering for ProM

Gabriel M. Veiga and Diogo R. Ferreira

IST – Technical University of Lisbon
Avenida Prof. Dr. Cavaco Silva
2744-016 Porto Salvo, Portugal
{gabriel.veiga,diogo.ferreira}@tagus.ist.utl.pt

Abstract. The goal of process mining is to discover process models from event logs. However, for processes that are not well structured and have a lot of diverse behavior, existing process mining techniques generate highly complex models that are often difficult to understand; these are called spaghetti models. One way to try to understand these models is to divide the log into clusters in order to analyze reduced sets of cases. However, the amount of noise and ad-hoc behavior present in real-world logs still poses a problem, as this type of behavior interferes with the clustering and complicates the models of the generated clusters, affecting the discovery of patterns. In this paper we present an approach that aims at overcoming these difficulties by extracting only the useful data and presenting it in an understandable manner. The solution has been implemented in ProM and is divided in two stages: preprocessing and sequence clustering. We illustrate the approach in a case study where it becomes possible to identify behavioral patterns even in the presence of very diverse and confusing behavior.

Key words: Process Mining, Preprocessing, Sequence Clustering, ProM, Markov Chains, Event Logs, Hierarchical Clustering, Process Models

1 Introduction

The main application of process mining is the discovery of process models. For processes with a lot of different cases and high diversity of behavior, the models generated tend to be very confusing and difficult to understand. These models are usually called *spaghetti models*. Clustering techniques have been investigated as a means to deal with this complexity by dividing cases into clusters, leading to less confusing models. However, results may still suffer from the presence of certain unusual cases that include noise and ad-hoc behavior, which are common in real-world environments. Usually this type of behavior is not relevant to understand a process and it unnecessarily complicates the discovered models.

In this paper we present an approach that is able to deal with these problems by means of sequence clustering techniques. This is a kind of model-based clustering that partitions the cases according to the order in which events occurred. For the purpose of this work the model used to represent each cluster is

a first-order Markov Chain. The fact that this clustering is probabilistic makes it suitable to deal with logs containing many different types of behavior, possibly non-recurrent behavior as well. When sequence clustering is applied, the log is divided into a number of clusters and the correspondent Markov Chains are generated. Additionally, the approach also comprises a preprocessing stage, where the goal is to clean the log of certain events that will only complicate the clustering method and its results. If after both techniques are applied the models are still confusing, sequence clustering can be re-applied hierarchically within each cluster until understandable results are obtained. This approach has been implemented in ProM [1], an extensible framework for process mining that already includes many techniques to address challenges in this area.

The paper is organized as follows: Section 2 provides an overview of existing work involving clustering and process mining. Section 3 presents the proposed approach, including the preprocessing stage and the sequence clustering algorithm. Section 4 demonstrates the approach in a real-world case study where the goal was to understand the typical behavior of faults in an application server. Section 5 concludes this paper.

2 Clustering in Process Mining

When generating process models, conventional control-flow mining techniques tend to over-generalize. In the attempt to represent all the different behavior present in the log these techniques create models that allow for more behavior than the one actually observed. When a log has process instances with very different behavior the generated models are even more complex and confusing. One way to address the problem is by means of clustering techniques [2].

One such approach has already been implemented in ProM and is known as the *Disjunctive Workflow Schema* (DWS) mining plug-in [3]. According to this methodology, first the complete log is examined and a model is generated using the *HeuristicsMiner* [4]. If the model generated is optimal and no over-generalization is detected the approach stops, otherwise the log is divided into clusters using the *K-means* clustering method. If the cluster models still allow for too much behavior the clusters are repartitioned and so on until optimal models are achieved.

Trace Clustering [5] is another technique implemented in ProM that aims at partitioning the log by grouping similar sequences together. The motivation for this technique is the existence of flexible environments, where the execution of processes does not follow a rigid set of rules. This approach makes use of *distance-based clustering* along with profiles, with the purpose of reducing the diversity and the complexity of models by lowering the number of cases analyzed at once. Each profile is composed by a set of features that describe and numerically classify a case from a particular perspective. Distance metrics (like the Euclidean distance or the Hamming distance) are then used to calculate the distance between two cases. Clustering methods such as *K-means Clustering* or *Self-Organizing Maps* (SOM) can then be used to group closely related cases

into the same cluster. Recent work in trace clustering includes the use of an edit distance between sequences, where the cost of edit operations can be determined by taking into account the context of an event within a sequence [6].

3 Sequence Clustering for ProM

Like the techniques described above, sequence clustering can take a set of sequences and group them into clusters, so that similar types of sequence are placed in the same cluster. But in contrast with the above techniques, sequence clustering is performed directly on the input sequences, as opposed to being performed on features extracted from those sequences. Sequence clustering has been extensively used in the field of bioinformatics, for example to classify large protein datasets into different families [7]. Process mining also deals with sequences, but instead of aminoacids the sequences contain events that have occurred during the execution of a given process. Sequence clustering techniques are therefore a natural candidate to perform clustering on workflow logs.

3.1 Sequence clustering

The sequence clustering algorithm used here is based on first-order Markov chains [8, 9]. Each cluster is represented by the corresponding Markov chain and by all the sequences assigned to it. For the purpose of process mining it becomes useful to augment the simple Markov chain model with two dummy states: the input and the output state. This is necessary in order to represent the probability of a given event being the first or the last event in a sequence, which may become useful to distinguish between some types of sequences. The use of input and output states is an extension to the work described in [10].

Figure 1 shows a simple example of such a chain depicted in ProM via the sequence clustering plug-in developed in this work. In this figure, darker elements (both states and transitions) are more recurrent than lighter ones. By analyzing the color of elements and the probability associated with each transition it is possible to decide which elements should be kept for analysis, and which elements can be discarded. For example, one may choose to remove transitions that have very low probabilities, so that only the most typical behavior can be analyzed. Although this kind of model is not as expressive as a Petri net, it can be useful to understand and apply post-processing to the generated cluster models.

The assignment of sequences to clusters is based on the probability of each cluster producing the given sequence. In general, a given sequence will be assigned to the cluster that is able to produce it with higher probability. Let ($\circ$) and ($\bullet$) denote the input and output states, respectively. To calculate the probability of a sequence $\overline{x} = \{\circ, x_1, x_2, \cdots, x_L, \bullet\}$ being produced by cluster c_k the following formula is used:

$$p\left(x \mid c_k\right) = p\left(x_1 \mid \circ; c_k\right) \cdot \left[\prod_{i=2}^{L} p\left(x_i \mid x_{i-1}; c_k\right)\right] \cdot p\left(\bullet \mid x_L; c_k\right) \tag{1}$$

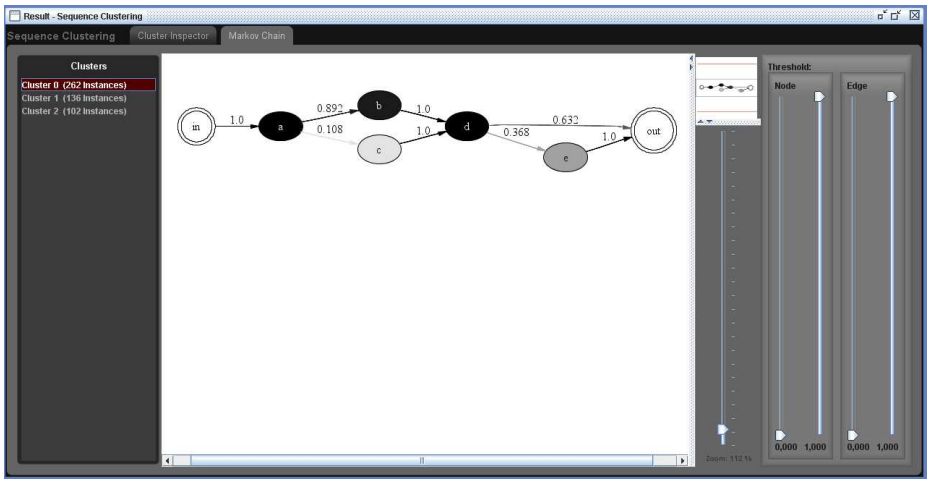

Fig. 1. Example of a cluster model displayed in the sequence clustering plug-in.

where $p(x_i \mid x_{i-1}; c_k)$ is the transition probability from x_{i-1} to x_i in the Markov chain associated with cluster c_k. This formula handles the input and output states in the same way as any other regular state that corresponds to an event.

The goal of sequence clustering is to estimate these parameters for all clusters c_k (with $k = 1, 2, \ldots, K$) based on a set of input sequences. For that purpose, the algorithm relies on an Expectation–Maximization procedure [11] to improve the model parameters iteratively. For a given number of clusters K the algorithm proceeds as follows:

1. Initialize randomly the state transition probabilities of the Markov chains associated with each cluster.
2. For all input sequences, assign each sequence to the cluster that can produce it with higher probability according to equation (1).
3. Compute the state transition probabilities of the Markov chain of each cluster, considering the sequences that were assigned to that cluster in step 2.
4. Repeat steps 2 and 3 until the assignment of sequences to clusters does not change, and hence the cluster models do not change either.

In other words, first we randomly distribute the sequences into the clusters (steps 1 and 2), then in step 3 we re-estimate the cluster models (Markov chains and their transition probabilities) according to the sequences assigned to each cluster. After this first iteration we re-assign the sequences to clusters and again re-estimate the cluster models (steps 2 and 3). These two steps are executed repeatedly until the algorithm converges. The result is a set of Markov models that describe the behavior of each cluster. In this work we have implemented this algorithm as a sequence clustering plug-in for ProM.

3.2 Applications of sequence clustering

Sequence clustering algorithms have been an active field of research in the area of bioinformatics [7, 12] and although this has been the area where it finds most applications, some work has been done with this type of algorithms in other areas as well. In [8], the authors analyze the navigation patterns on a website, where these patterns consisted of sequences of URL categories visited by users. Sequence clustering was used to identify common user profiles by placing users with similar navigation paths in the same cluster.

Sequence clustering has also been used in the field of process mining [10], where the authors described two experiments of identifying typical behavior. One experiment used an event log collected manually from the activities of a software development team, and allowed the discovery of the typical interaction patterns between members of that team. The other experiment was conducted over traces collected from a datavase system in order to identify common routines. In both experiments the authors made use of the sequence clustering algorithm implemented in Microsoft SQL Server [13].

3.3 Preprocessing

Although the sequence clustering algorithm described above is robust to noise, all sequences must ultimately be assigned to a cluster. If a sequence is very uncommon and different from all the others it will affect the probabilistic model of that cluster and in the end will make it harder to interpret the model of that cluster. To avoid this problem, some preprocessing must be done to the input sequences prior to applying sequence clustering. This preprocessing can be seen as a way to clean the dataset of undesired events and also a way to eliminate undesirable sequences. For example, undesired events can be those that occur only very rarely, and undesired sequences can be single-step sequences that have only one event.

Some of the steps that can be performed during preprocessing are described in [9] and include, for example, dropping events and sequences with low support. In this work we have extended these steps by allowing not only the least but also the most recurring events and sequences to be discarded. This was motivated by the fact that in some real-world applications the log is filled with some very frequent but unrelated events that must be removed in order to allow the analysis to focus on the relevant behavior. Spaghetti models are often cluttered with events that occur very often but only contribute to obscure the process model one aims to discover.

The preprocessing steps implemented within the sequence clustering plug-in are optional and configurable. They focus on the following features:

1. *Event type* – The events recorded in a MXML log file [14] may represent different points in the lifetime of workflow activities, such as the start or completion of a given activity. For sequence clustering what is important is the order of activity execution, so we retain only one type of event and that

is usually the completion event for each activity. Therefore only events of type "complete" are kept after this step.
2. *Event support* – Some events may be so infrequent that they are not relevant for the purpose of discovering typical behavior. These events should be removed in order to facilitate analysis. On the other hand, some events may be so frequent that they too became irrelevant and even undesirable if they hide the behavior one aims to discover. Therefore, this preprocessing can remove events both with too low and too high support.
3. *Consecutive repetitions* – Sequence clustering is a means to analyze the transitions between states in a process. If an event is followed by an equal event then it should be considered only once, since the state of the process has not changed. Consecutive repetitions are therefore removed, for example: the sequence $A \rightarrow C \rightarrow C \rightarrow D$ becomes $A \rightarrow C \rightarrow D$.
4. *Sequence length* – After the previous preprocessing steps, it may happen that some sequences collapse to only a few events or even to a single event. This preprocessing step provides the possibility to discard those sequences. It also provides the possibility to discard exceedingly long sequences which can have undesirable effects in the analysis results. Sequence length can therefore be limited to a certain range.
5. *Sequence support* – Some sequences may be rather unique so that they hardly contribute to the discovery of typical behavior. In principle the previous preprocessing steps will prevent the existence of such sequences at this stage but, as with events, sequences that occur very rarely can be removed from the dataset. In some applications such as fault detection it may be useful to actually discard the most common sequences and focus instead on the less frequent ones, so sequence support can also be limited to a certain range.

The order presented is the order in which the preprocessing steps should be applied, because if the steps are applied in a different order the results may differ. For example, rare sequences should only be removed at the final stage, because previous steps may transform them into common sequences. Imagining we have the rare sequence $A \rightarrow B \rightarrow C \rightarrow D$, but in step 2 state B is considered to have low support and is removed, then it becomes $A \rightarrow C \rightarrow D$. This new sequence might not be a rare sequence and therefore should not be removed.

3.4 Implementation within ProM

The above preprocessing steps and the sequence clustering algorithm have been implemented and are available as a new plug-in for the process mining framework ProM[1]. Figure 2 shows the inputs and outputs for this plug-in.

The preprocessing stage receives an input log in MXML format [14] and also some options provided by the user, which specify the parameters to be used in the preprocessing steps described above. The result is a filtered log. This log is made available to the ProM framework, so that it may be analyzed with other

[1] The ProM framework can be found at `http://prom.win.tue.nl/tools/prom`

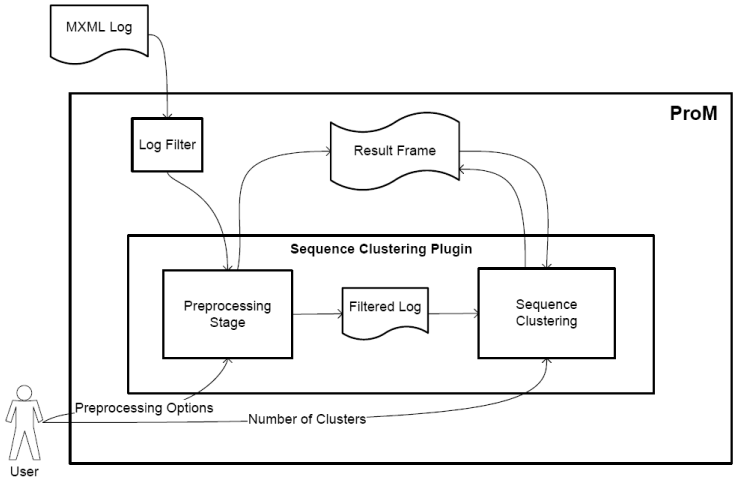

Fig. 2. Sequence Clustering plug-in in the ProM framework

plug-ins if desired. Instead of acting just as a first stage to sequence clustering, the preprocessing stage can also be used together with other types of analysis available in the framework. Figure 3 presents a screenshot of this stage, depicting the options available for the user to pre-process the input log (top-right corner).

Fig. 3. Preprocessing stage for the Sequence Clustering plug-in

The sequence clustering stage receives the filtered log as input from the preprocessing stage and also the desired number of clusters. In general the plug-in

will generate a solution with the provided number of clusters except when some clusters turn out to be empty. The plug-in provides special functionalities for visualizing the results, both in terms of sequences that belong to each cluster and in terms of the Markov chain for each cluster. Each cluster can be used again as an event log in ProM, so it becomes possible to further subdivide it into clusters, or for another process mining plug-in to analyze it. These features allow the user to drill-down through the behavior of clusters.

On one hand, sequence clustering is a mining plug-in that extracts models of behavior for the different behavioral patterns found in the event log. Figure 1 shows the type of results that the plug-in is able to present. When visualizing the results, the user can adjust thresholds that correspond to the minimum and maximum probability of both edges and nodes (right-hand side of fig.1). This allows the user to adjust what is shown in the graphical model by removing elements (both states and transitions) that are either too frequent or too rare. This feature facilitates the understanding of spaghetti models without having to re-run the algorithm again.

On the other hand, sequence clustering can also be regarded as an analysis plug-in since it generates new events logs that can be analyzed by other plug-ins available in the ProM framework. This is also useful for analyzing spaghetti models, which are hard to understand at first, but can be made simpler by dividing their complete behavior into a set of clusters that can be analyzed separately by other algorithms.

4 Case Study: Application Server Logs

Public administration and public services often have large-scale IT systems that serve thousands of users. These systems are usually backed by an infrastructure that involves replication, redundancy and load balancing. Due to the large number of replicated software applications and due to the large number of simultaneously connected users, it becomes exceedingly difficult to determine the cause for some malfunctions that produce instabilities that propagate across the system and negatively affect the experience of several users at the same time.

In this section we present one such case study based on the experience at a public institution. At the time the institution was struggling with complaints about a situation in which the applications would freeze or crash unexpectedly for several users at the same time. The applications are Java-based and were developed according to a client/server architecture where the end users had a fat client and the back-end was implemented as a set of Enterprise JavaBeans hosted in an application server that has been replicated across a server farm.

Since the root cause for this malfunction was hard to determine, we turned to the application server logs in order to study the exceptions that had been recorded for each Java thread. This proved to be quite difficult, not only for the overwhelming amount of exceptions being recorded all the time, but also for the fact that it was difficult to establish any causal relationship between

those exceptions. Figure 4 depicts the result of a first attempt to analyze the application server logs using the heuristics miner [4].

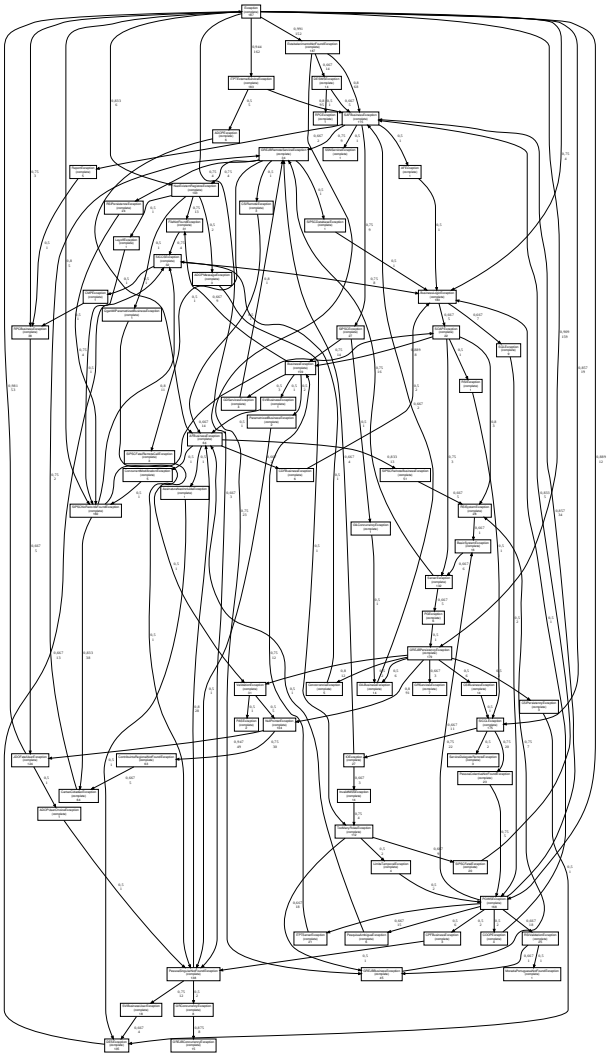

Fig. 4. Spaghetti model obtained from the application server logs using the heuristics miner.

Using the sequence clustering plug-in and its preprocessing capabilities, as well as the possibility of visually adjusting the cluster models according to certain thresholds, it was possible to identify several patterns involving different types of exceptions. These patterns were found after several attempts of tuning with the preprocessing parameters, selecting the number of clusters (typically from 3 to

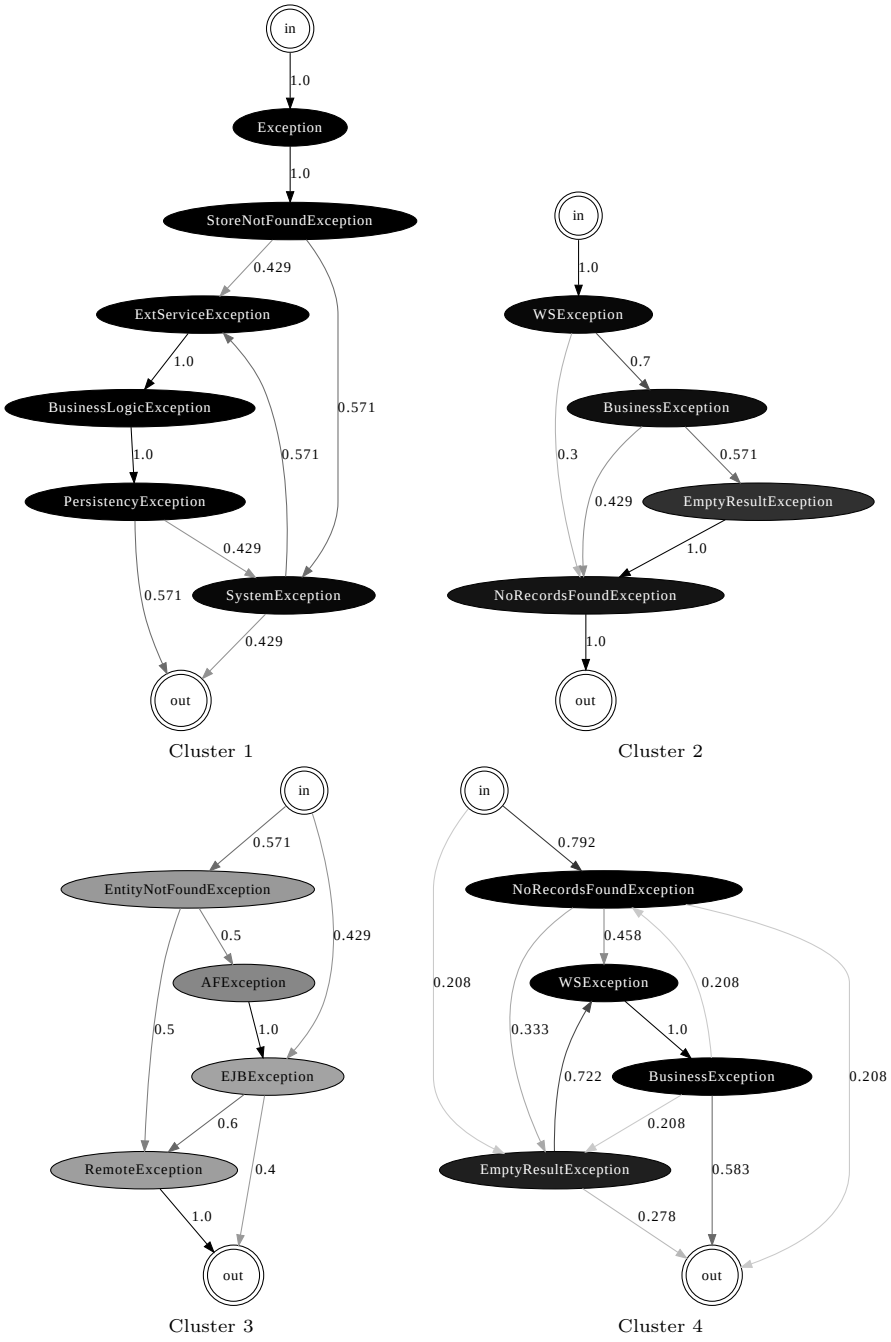

Fig. 5. Some of the behavioral patterns discovered from the application server logs using the sequence clustering plug-in.

12) and applying thresholds to the cluster models in order to make them more understandable. Figure 5 shows a selection of four clusters from the analysis results. Each of these clusters represents about 10% of the sequences in the original event log.

Cluster 1 has to do with a condition where a given element that was being searched in the database was not found. Usually, the user input is validated prior to querying the database, so it was assumed that the search would always produce some result. However, when it happened that the element being searched for was actually not found in the database, this triggered different kinds of system exceptions. This model helped identify the root cause for those exceptions.

Clusters 2 and 4 depict the two sides of what was thought to be a single relationship between web service exceptions and queries that return empty result sets. In cluster 2 we have the situation in which the application invokes a web service that fails to insert data and this leads to errors in application code that cannot find the data afterwards. In cluster 4 the application is invoking a web service that queries the database; in this case the web service fails because no records were found in the first place. This model helped realize that there were actually two different problems involving these exceptions.

Cluster 3 had to do with platform exceptions that occurred when the system was overloaded during peaks of activity. In this case the system was unable to find application objects due to the lack of resources. This problem was mitigated by adjusting the parameters of some application server components.

These and other behavioral patterns found via the sequence clustering plug-in contributed to focusing the effort on the exceptions that were the most likely cause for the observed problems. Otherwise, it would have been very difficult to identify the relevant events amidst an overwhelming amount of data in the application server logs. Besides the sequence clustering plug-in, it should be noted that other functionalities available within the ProM framework – such as the log summary and basic log filtering – have been very useful during the first stages of log preprocessing.

5 Conclusion

Understanding the run-time behavior of business processes is made difficult by the fact that real-world processes often involve a significant amount of unstructured and ad-hoc behavior, which produces confusing, spaghetti-like models. To address this problem we have developed and presented in this paper a solution that employs preprocessing techniques and that is based on a sequence clustering algorithm. This becomes a very helpful technique to discover behavioral patterns and to visualize them, since its probabilistic nature is inherently able to deal with noise and to separate different behaviors into a set of cluster models. The approach has been implemented as a plug-in for the ProM framework and has been applied in a real-world case-study to detect faulty behavior in a large-scale application server. Presently, we are evaluating the results of sequence clustering in comparison with other clustering methods based on fitness metrics.

References

1. van Dongen, B., de Medeiros, A.A., Verbeek, H., Weijters, A., van der Aalst, W.: The ProM framework: A new era in process mining tool support. In Ciardo, G., Darondeau, P., eds.: Application and Theory of Petri Nets 2005. Volume 3536 of Lecture Notes in Computer Science., Springer-Verlag, Berlin (2005) 444–454
2. Greco, G., Guzzo, A., Pontieri, L., Saccà, D.: Mining expressive process models by clustering workflow traces. In Dai, H., Srikant, R., Zhang, C., eds.: PAKDD. Volume 3056 of Lecture Notes in Computer Science., Springer (2004) 52–62
3. de Medeiros, A.K.A., Guzzo, A., Greco, G., van der Aalst, W.M.P., Weijters, A.J.M.M., van Dongen, B.F., Saccà, D.: Process mining based on clustering: A quest for precision. In ter Hofstede, A.H.M., Benatallah, B., Paik, H.Y., eds.: Business Process Management Workshops. Volume 4928 of Lecture Notes in Computer Science., Springer (2007) 17–29
4. Weijters, A., van der Aalst, W., de Medeiros, A.A.: Process mining with the heuristicsminer algorithm. BETA Working Paper Series WP 166, Eindhoven University of Technology (2006)
5. Song, M., Gnther, C., van der Aalst, W.: Trace clustering in process mining. In: Proceedings of the 4th Workshop on Business Process Intelligence (BPI'08), BPM Workshops 2008, Milan (September 1, 2008)
6. Bose, R.P.J.C., van der Aalst, W.M.P.: Context aware trace clustering: Towards improving process mining results. In: SDM, SIAM (2009) 401–412
7. Enright, A.J., Ouzounis, C.: Generage: a robust algorithm for sequence clustering and domain detection. Bioinformatics **16**(5) (2000) 451–457
8. Cadez, I., Heckerman, D., Meek, C., Smyth, P., White, S.: Model-based clustering and visualization of navigation patterns on a web site. Data Mining and Knowledge Discovery **7**(4) (2003) 399–424
9. Ferreira, D.: Applied sequence clustering techniques for process mining. In Cardoso, J., van der Aalst, W., eds.: Handbook of Research on Business Process Modeling. IGI Global (2009)
10. Ferreira, D., Zacarias, M., Malheiros, M., Ferreira, P.: Approaching process mining with sequence clustering: Experiments and findings. In: Proceedings of the 5th International Conference on Business Process Management (BPM 2007). Volume LNCS 4714., Springer (2007) 360–374
11. Dempster, A., Laird, N., Rubin, D.: Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society, Series B **39**(1) (1977) 1–38
12. Enright, A.J., van Dongen, S., Ouzounis, C.: An efficient algorithm for large-scale detection of protein families. Nucleic Acids Research **30**(7) (2002) 1575–1584
13. Tang, Z., MacLennan, J.: 8. In: Data Mining with SQL Server 2005. Wiley Publishing, Inc. (2005) 209–227
14. van Dongen, B., van der Aalst, W.: A meta model for process mining data. In Casto, J., Teniente, E., eds.: Proceedings of the CAiSE'05 Workshops (EMOI-INTEROP Workshop). Volume 2., FEUP, Porto, Portugal (2005) 309–320