

An efficient SOM-based pre-processing to improve the discovery of frequent patterns in alarm logs

F. Fessant

France Telecom Research and Development
FTR&D/TECH/SUSI/TSI
2 av. P. Marzin, 22307 Lannion, France

F. Clérot

France Telecom Research and Development
FTR&D/TECH/SUSI/TSI
2 av. P. Marzin, 22307 Lannion, France

Abstract - *We describe a pre-processing technique for mining a telecommunication alarm log for frequent temporal patterns. The method consists in extracting relevant subsets from the initial log with the aim of discovering frequent patterns more accurately. In a first step, the alarm types presenting the same temporal behaviour are clustered with a self organizing map. Then, log areas which are rich in alarms of each cluster are regrouped in subsets which are subsequently exhaustively searched for frequent patterns. We demonstrate the efficiency of our pre-processing method through experiments on an actual alarm log from an ATM network.*

Keywords: Self-organizing map, chronicle recognition, telecommunications network supervision.

1 Introduction

Telecommunications networks are growing in size and complexity, which means that a constantly increasing volume of notifications must be handled by the management system. Most of this information is produced spontaneously by equipments (e.g. status change and dysfunction detection) and this message flow must be preprocessed to make an effective management possible. Filters based on a per-notification basis fail to perform an adequate information pre-processing required by human operators or by management application software which are not able to process such amount of events. A pre-processing stage must “thin” this information stream by suppressing redundant notifications and/or by aggregating relevant ones. Numerical time constraints must also be taken into account since time information is apropos for the telecommunications alarm propagation. Many works deal with different approaches and propose more or less complex intelligent filtering: one can use some efficient rule-based languages [1], and/or object-based techniques [2]. More specific techniques are devoted to capture time

constraints between alarms [3], [4]. In any case, the problem of expertise acquisition remains the same: how to feed the filtering system? Which aggregation rules are relevant? A way to filter the information flow is to exploit the logs collected from telecommunications equipment. We have developed a tool called FACE (Frequency Analyzer for Chronicle Extraction) which performs a frequency-based analysis in order to extract “frequent patterns” from the logs. We only suppose that all the events are time-stamped. The searched patterns are sets of event patterns with time constraints between them (these sets are called “chronicle models”) and the frequency criterion is defined as a user-defined minimal frequency threshold [5]. Identifying the most frequent chronicle models is relevant to reduce the number of alarms displayed to the operator: if a chronicle corresponds to a dysfunction, the corresponding set of alarms is aggregated before being displayed to the human operator; if not, the corresponding set of alarms is filtered. At the moment, we do not use any extra knowledge about the domain; the rule qualification (aggregation or filtering) is performed by an expert at the end of the discovering process. Real experiments on telecommunications data show that most of the discovered chronicles are relevant and some of them have a real benefit for the experts. However, the chronicle process implemented in FACE, based on the exhaustive exploration of the chronicle instances in the alarm log, is very memory-space consuming and the main factor of this explosion is the size of the processed event logs. To deal with that problem in the current implementation of FACE, the operational experts manually select some alarms types and/or some time periods in the alarms logs in order to extract a more manageable sublog to be processed by the tool. The purpose of this communication is to describe and to evaluate a pre-processing method to automatically extract relevant sublogs from an initial alarm log to simplify the use of the software and alleviate the memory saturation effect. The pre-processing stage we propose can be decomposed in two main steps. Firstly we group together the alarm types which exhibit the same temporal behavior. We introduce an original

parameterization scheme to accurately represent the alarms with a reduced set of parameters. Then we search for each group the areas of the log that are rich in alarms of the observed group. The sublogs are built from the alarms in these areas and the frequent patterns are searched in sublogs. The following sections describe our data pre-processing and the experimental results obtained on an actual alarm log from an ATM network.

2 Chronicle discovery with FACE

In this section we briefly introduce several definitions and we present FACE learning algorithm.

2.1 Definitions

A *chronicle model* is a set of events linked together with time constraints.

A *chronicle instance* is a set of events of the log whose occurrence dates respect the time constraints.

We define the *frequency* of a chronicle model as the number of its instances in the given log. A chronicle model is *frequent* if its frequency is greater than a user defined threshold n_{qmin} .

2.2 FACE algorithm

The main idea of the FACE algorithm is that a chronicle model won't be frequent if one of its sub-chronicles is infrequent (a sub-chronicle of a chronicle is a chronicle with a subset of alarms of the chronicle). Otherwise, if all the sub-chronicles of a chronicle are frequent, the chronicle may be frequent. The intuitive solution to compute the set of candidates of size n (with n events) is to generate all the chronicles of size n from one of its frequent sub-chronicle of size $n-1$ and then to check if all their sub-chronicles are frequent. FACE algorithm starts by computing the set of frequent chronicles of size $n=1$, which corresponds to the set of all the frequent alarms in the alarms log. It then builds iteratively the exhaustive set of all frequent chronicles. At iteration n , it first generates the set of candidates of size n from the set of frequent of size $n-1$. Then, a Chronicle Recognition System [3] is used to enumerate the instances of the candidates and to calculate the frequency of each candidate. FACE keeps only the frequent ones for the next step. The main loop ends at the iteration n when there is no frequent chronicle of size n . The user must set another parameter: the time window t_w which sets the maximum distance between the events of a chronicle. More details are available in [5]. Remember that the qualification (relevant or not) of chronicles discovered by FACE is not our purpose. This qualification must be performed by the domain monitoring experts at the end of the discovery process.

3 Data representation

Previous experiments with FACE on alarm logs of various origins have revealed that the instances of chronicles tend to be distributed in a clustered way along time. This can be explained by the fact that most chronicles are indicative of faulty network states, which do not happen randomly in time but only happen for short periods of time before being corrected. The data representation introduced below takes this into account, gives a great importance to the temporal behavior of the alarms and aims at grouping in sublogs alarms with similar temporal behaviors. Alarms of different sublogs are supposed to belong to different chronicles.

3.1 Cumulative profiles of alarms

The alarm types are represented through their temporal evolution along the log by their cumulative profiles. The cumulative profile of an alarm is computed by adding up its successive occurrences and it is normalized in order to make it independent of the total number of occurrences. Our experiments are achieved with ATM (Asynchronous Transfer Mode) network data. We have about one-month alarms log with 46600 alarms dispatched through 12160 different types. These alarms are indicative of network hardware crashes. An alarm type is characterized by an alarm message (a string) and a date and time of occurrence; 75% of the alarm types weight less than 3 occurrences. Table 1 shows an extract of the log with 4 successive typical alarm messages, referenced by their date and time of occurrence.

Table 1: extract of the alarm log

date	time	Alarm message
3/10/91	9:19:15	nsarc102 OV_Node_Major 4 nsarc102: Plusieurs interfaces du noeud nsarc102 sont tombees
3/10/91	9:19:17	nsrou102 OV_Node_Up 1 nsrou102: Node up
3/10/91	9:20:39	nsarc106 OV_Node_Down 5 nsarc106: Node down
3/10/91	9:21:27	nsmon203 sdsols_interface_ko 4 Au moins une interface est KO (Trap Sdsols specific 23)

Fig. 1 shows the occurrences of two distinct alarm types and Fig. 2 shows the corresponding normalized cumulative profiles. The 806 occurrences of the alarm type 1 occur only in the beginning of the log; the 73 occurrences of alarm type 2 are more regularly distributed on the observation period.

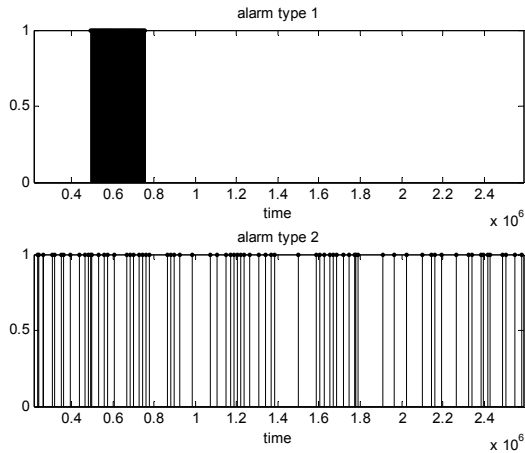


Fig. 1 Occurrences of two distinct alarm types of the log. Occurrences of alarm type 1 occur only in the beginning of the log (up), those of alarm type 2 are more regularly distributed on the observation period (bottom).

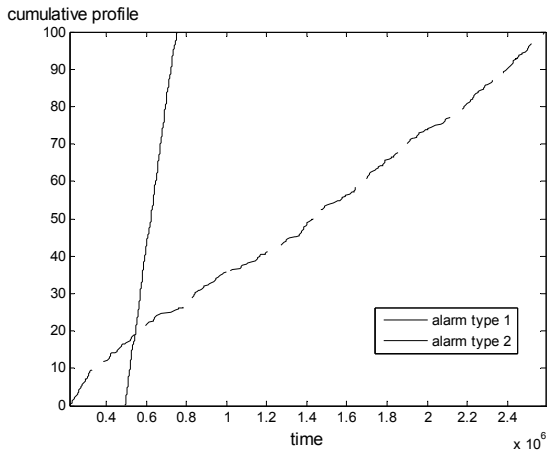


Fig. 2 Normalized cumulative profiles of the alarm types of Fig. 1.

The shape of a cumulative profile informs about the accumulation behavior of the alarm: the sharper the rise of a cumulative profile, the stronger the accumulation of the alarms. The shapes can be compared together: similar shapes correspond to alarms with close temporal evolution.

3.2 Parameterization of cumulative profiles

The cumulative profile of an alarm is parameterized with a limited number of descriptors. We estimate the time instants of the log when $x\%$ of the occurrences of the alarm are obtained (hereafter, x varies from 1 to 100 by step $\Delta x = 1$, but other choices are possible). The time instants are estimated from real

cumulative profiles by linear interpolation. Fig. 3 illustrates the parameterization process with an example. We plot one of the real cumulative profiles of Fig. 2 and its estimated profile. We focus on two of the descriptors: time instant for which 25% of the occurrences of the alarm type are obtained (t_{25}) and time from which we reach 50% of the occurrences (t_{50}).

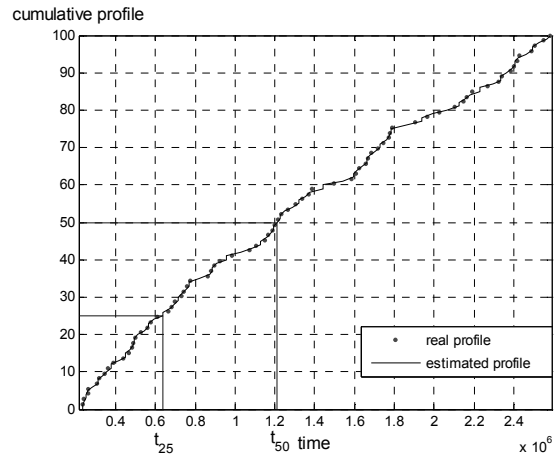


Fig. 3 Estimated and real cumulative profiles of an alarm.

From now all the alarm types are described with the same number of parameters (100 in this case) whatever their cumulative profiles.

The benefit of such parameterization of the profile is to make it easy to tune and independent of the log duration. The simple alternative parameterization of [6] which just records how many events fall in successive time intervals of duration Δt implies to strike a delicate balance between the duration Δt and the size of the parameterization: too large a value of Δt will not allow a fine description of the temporal behavior of the events and too small a value may result in a very large vector to describe an event for logs with long durations. This balance is all the more difficult to strike as different events may have very different relevant accumulation time scales so that a good choice of Δt for some events may be wrong for others. The parameterization scheme introduced above can be applied whatever the log duration and allows the representation of data with a highly reduced set of parameters. Moreover, it is naturally adapted to the possibly different dynamics of the alarms as the representation is deduced from the accumulation profiles themselves and not from a projection of these profiles on time intervals of a priori determined duration Δt . Therefore, the proposed parameterization allows to finely describe the accumulation profile of each alarm while staying independent of the log duration.

The granularity level Δx can be set by the user; we have found $\Delta x = 1$ to be fine enough to allow an accurate description of the profiles in our application.

The parameters for an alarm type with regular accumulation will be regularly spaced in time along the log, those for an alarm type with strong accumulation will be very close and concentrated in a restricted time period of the log.

The purpose of the next step of the method is to group together the alarm types, described by their estimated cumulative profiles, which have the same temporal behavior.

4 Clustering of estimated profiles

4.1 Clustering with a Self Organizing Map

The estimated profiles are grouped together with a Self Organizing Map (SOM). A SOM is a useful tool for exploratory data analysis made up of a set of nodes organized into a 2-dimensional grid (the *map*). Each node has fixed coordinates in the map and adaptive coordinates (the *weights*) in the input space. The input space is relative to the variables setting up the observations. The self-organizing process slightly moves the nodes coordinates in the data definition space -i.e. adjusts weights according to the data distribution. This weight adjustment is performed while taking into account the neighboring relationships between nodes in the map. The SOM has the well-known ability that the projection on the map preserves the proximities: close observations in the original multidimensional input space are associated with close nodes in the map. For a complete description of the SOM properties and some applications, see [7] and [8]. After learning has been completed, the map is segmented into clusters, each cluster being formed of nodes with similar behavior, with a hierarchical agglomerative clustering algorithm. This segmentation simplifies the quantitative analysis of the map [9].

4.2 Clustering results

We consider only the frequent alarm types -i.e. alarm types making at least 3 occurrences in the log: a chronicle being by definition characterized by instances with several occurrences in the log, infrequent alarm types cannot take part in the chronicle creation. We end up with a database of 3042 cumulative profiles (the observations) described on the space of the 100 descriptors. We experiment with a 9x9 square map with hexagonal neighborhoods (experiments on SOM have been done with the SOM Toolbox package for Matlab [10]).

The clustering of the map has revealed 10 different groups of observations with similar behavior.

Fig. 4 details the characteristic behavior of the clusters. We plot the "mean" cumulative profile for each of the 10 clusters; each of the clusters being identified by a number. The "mean" cumulative profile of a cluster is computed by the mean of all the observations that have been classified by the nodes of the map belonging to the cluster. We weight each observation by its occurrences number to strengthen the weight of alarm types with many occurrences. The visual inspection of the figure shows that the clusters indeed correspond to very different accumulation behaviors: some clusters correspond to accumulations on rather limited sets of periods (cluster 1, accumulating at the very beginning of the log, cluster 7 accumulating at the very end or cluster 9 accumulating in the middle, for instance); other clusters show a more even distribution in time (clusters 5 and 6, for instance).

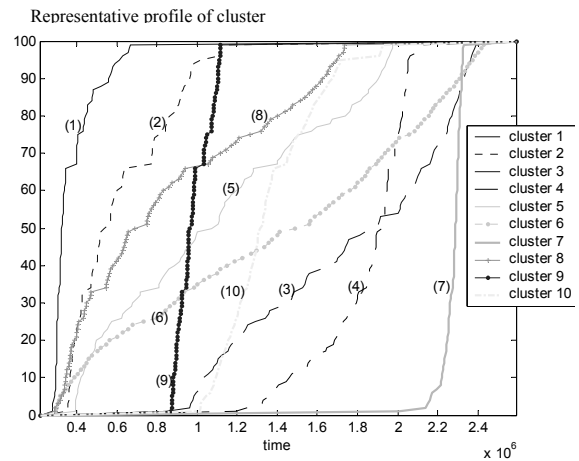


Fig. 4 "Mean" cumulative profiles of the clusters.

At this stage we end up with a limited number of groups that characterize and summarize the whole alarm types: all the alarms types classified in a cluster accumulate in the same way in the log.

5 Sublog construction

Some areas of the log are more significant than others for a given cluster of alarm types. We proceed as follows to identify these areas. For each cluster, the log is split in several slices of variable duration; a slice being delimited by two successive parameters (i.e. two successive time instants) of the mean cumulative profile of the cluster. The slices induced by one cluster form a partition of the log. They constitute the base from which each sublog is built. We define the importance of a slice for a cluster by the number of alarms of the cluster included in the slice, in proportion of all the alarms this slice contains (with this definition, we tend to favor some slices producing few alarms but essentially alarms

of the cluster). We now select the most interesting slices for a cluster: we first order the slices according to their importance for the studied cluster; the slices which have the highest importance values are ranked first. The subset of slices corresponding to the cluster is simply built by adding the slices in order of decreasing importance until a given proportion of the alarms of the cluster are contained in the subset. Fig. 5 illustrates the result of the slice selection process for cluster 1. We arbitrarily fixed the stopping criterion to 90% (i.e. we retain the slices until the sublog contains 90% of the alarms of the cluster). For the cluster given here as example in the figure, we select 82 slices of the log and this corresponds to 22% of the whole alarms. A black rectangle on the x-axis marks the selected slices.

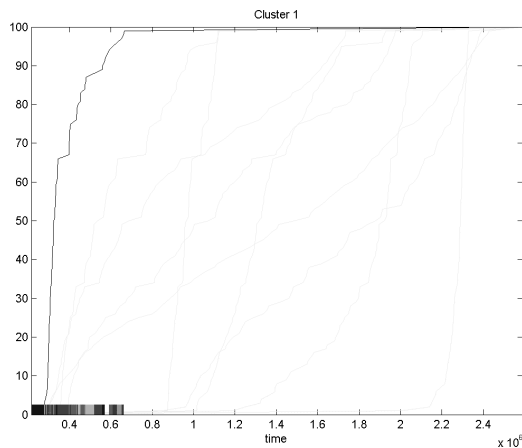


Fig. 5 Selected areas of the log for cluster 1.

The selected areas clearly coincide with the accumulation areas of the cluster (essentially in the beginning of the log for the given example). Fig. 6 plots the location of the selected slices in the log for each cluster. The comparison with Fig. 4 shows that the selected areas for a given cluster correspond to the areas of alarm accumulation for this cluster.

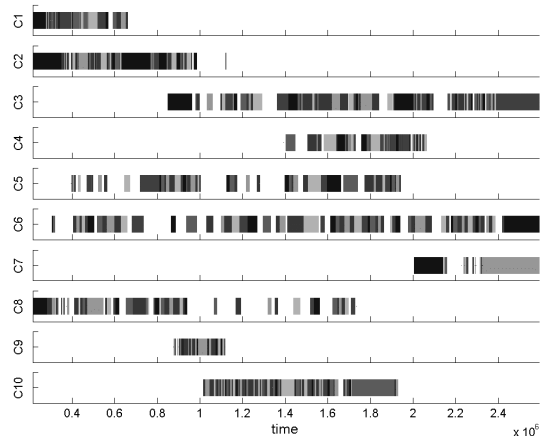


Fig. 6 Time location of selected areas for the clusters.

At this step of the process, we have isolated as many subsets as alarm type clusters. The sublog attached to a cluster of alarms is simply obtained by re-ordering in time the alarms of the corresponding subset. Note that we keep all the alarms belonging to the slices, including those of infrequent alarm types. The table 2 gives the number of slices in each subset and the corresponding number of alarms.

Table 2: Number of slices and alarms in each sublog

Sublog index	Number of slices	Number of alarms
1	82	10393
2	79	15562
3	70	15159
4	69	5526
5	54	12309
6	64	16149
7	32	11519
8	50	14796
9	71	3979
10	88	12994
Total		118386

Let us remark that our selection method considers each cluster independently of the others and, therefore, a same time area can be assigned to several subsets. The effect is that the sum of alarms over the all sublogs is about 2.5 the number of alarms of the initial log (recall that the entire log consists of 46600 alarms).

6 Experimental results

In this section we report the experimental results we obtained when running the FACE algorithm on the

initial log and on the sublogs designed as previously explained (from now on we call them SOM sublogs). The FACE learning algorithm requires two input parameters: the time window t_w which sets the maximum distance between the events of a chronicle model and the minimum threshold of the chronicle instances n_{qmin} which gives the minimum number of instances a chronicle model must have in the log to be considered as frequent. Time window t_w is fixed to 15 seconds and we choose to vary the value of n_{qmin} .

We report on Fig. 7 the number of different chronicle models discovered on the entire log and on the SOM sublogs as a function of n_{qmin} . We also plot this number for normal sublogs. To build normal sublogs we simply manually split the whole log into several successive slices with about the same number of alarms in each slice. We proceed as operational experts usually do to extract more manageable sublogs from an alarm log. We have experimented with four different sets of sublogs (the log has been successively split in 2, 3, 5 and 10 sublogs.). The highest number of chronicles discovered with normal sublogs was for the set of 3 sublogs. In the scope of this paper we consider only the best result. More details can be found in [6]. All experiments reported below have been run on the same computer (a Pentium 4 with 1.7 GHz of CPU and 1 Go of RAM) with no other application running than FACE.

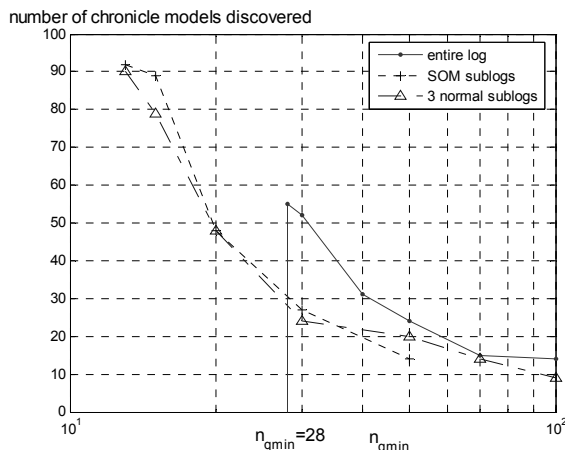


Fig. 7 Number of chronicle models discovered.

We can observe that the entire log is impossible to process for n_{qmin} below 28; the reason is a lack of memory-space of the computer. The treatment remains possible on sublogs for much smaller values of n_{qmin} (the limit is reached for $n_{qmin} = 13$ for the sets of sublogs, and it was never possible to go below this value).

We are able to discover more chronicles with the sublogs than with the entire log: for a given n_{qmin} , we discover more chronicles with the entire log than with the sublogs, but the exploration can be carried on

sublogs for much lower n_{qmin} . Moreover, a detailed analysis of the discovered chronicles shows that we find from the sublogs all the chronicles we have found from the entire log and also many new chronicles: we do not lose any chronicle with SOM sublogs.

We now compare the number of discovered chronicle models for low values of n_{qmin} , with normal sublogs and with SOM sublogs. SOM sublogs show a better performance as we found more chronicles with SOM sublogs as compared to the best set of manual sublogs. Therefore, the improvement obtained with the SOM sublogs cannot be attributed to the sequential processing of many smaller logs instead of the entire log and to the ability to reach low frequencies. This improvement has to be attributed to the data processing method adopted to build relevant sublogs. Let us remark that the manual search of the good number of sublogs that will lead to the discovery of the maximum number of chronicles is a very time consuming process. The processing method we propose automatically extracts the relevant sublogs.

These experimental results validate our data processing method which takes into account the fact that chronicle instances do not happen randomly in time but in a rather clustered way; using this pre-processing we recover all the chronicle models discovered without pre-processing and we can analyze the log in a more accurate way and discover new chronicles. We also have observed that the total processing time of the SOM sublogs remains of the same order of magnitude as for the entire log.

7 Conclusion

The system FACE is very helpful for experts to discover monitoring knowledge from alarm logs. However, the chronicle discovery process implemented in FACE (based on the exploration of all the possibilities of chronicle instances) has the limitation to be very memory-space consuming; the main factor of this explosion being the size of the alarm log. In this paper we have proposed a method to automatically extract relevant sublogs from an alarm log to simplify the use of the software and alleviate the memory saturation effect.

The method is decomposed in several steps. The first step consists in the description of the alarm types in a suitable representation that takes into account the temporal evolution of the alarms through the log. The alarms are described with very few parameters in the time space. The interest of our representation mode is to accurately describe the temporal behavior of each alarm while being independent of the log duration. Then the alarm types which have the same temporal behavior are grouped together with a self-organizing map. The result of the clustering is a description of the whole alarm

types in few groups. Finally, the areas of the log that are the more relevant for the clusters of alarm types are isolated and the alarms in these areas are grouped in sublogs. The sublogs can be processed with FACE.

We presented experiments on an actual ATM log with alarms of network hardware crashes. The proposed data processing method with its alarm representation scheme turns out to be very effective: first, we have recovered from the sublogs all the chronicles that we were able to find while processing the whole log. Moreover we can carry the search on sublogs to a point that is impossible to reach with the entire log and obtain many new chronicles. The performance is also better than the best performance reached with a costly manual pre-processing of the log. We also have observed that the total duration process with the sublogs always remains moderate. Another interesting point is that the user can get intermediate results after the processing of each sublog and future research will consider the adaptation of the learning parameters of the tool independently for each sublog in order to discover the maximum number of chronicles. The method can be applied on various types of logs like logs of attacks attempts so long as alarms can be temporally correlated.

8 References

- [1] M. Möller, S. Tretter and B. Fink, "Intelligent filtering in network management systems," in *4th Internal Symposium on Integrated Network Management*, 304-315, 1995.
- [2] Y.A. Nygate, "Event correlation using rule and object based techniques," in *4th Internal Symposium on Integrated Network Management*, 279-289, 1995.
- [3] C. Dousson, "Extending and unifying chronicle representation with event counters," in *15th ECAI*, 257-261, 2002.
- [4] G. Jakobson and M. Weissman, "Real time telecommunication network management: extending event correlation with temporal constraints," in *4th Internal Symposium on Integrated Network Management*, 290-301, 1995.
- [5] C. Dousson and T. Vu Duong, "Discovering chronicles with numerical time constraints from alarm logs for monitoring dynamic systems," in *16th IJCAI*, 620-626, 1999.
- [6] F. Fessant, F. Clérot and C. Dousson, "Mining of an alarm log to improve the discovery of frequent patterns," in *4th Industrial Conference on Data Mining*, 2004.
- [7] T. Kohonen, *Self Organizing Maps*, New York: Springer-Verlag, 3rd edition, 2001.
- [8] E. Oja and S. Kaski, *Kohonen Maps*, Elsevier, 1999.
- [9] J. Vesanto and E. Alhoniemi, "Clustering of the self-organizing map," *IEEE Trans. Neural Networks*, vol. 11 (3), pp. 586-600, 2000.
- [10] J. Vesanto, J. Himberg, E. Alhoniemi and J. Parhankangas, "SOM Toolbox for Matlab," Report A57 Helsinki University of Technology, Neural Network Research Centre, Espoo, Finland, 2000.