Buletinul Ştiinţific al Universităţii "Politehnica" din Timişoara

Seria ELECTRONICĂ şi TELECOMUNICAŢII
TRANSACTIONS on ELECTRONICS and COMMUNICATIONS

Tom 51(65), Fascicola 2, 2006

Application for Frequent Pattern Recognition in Telecommunication Alarm Logs

Petru Serafin[1], Alimpie Ignea[2]

Abstract – Based on an algorithm for frequent pattern recognition, this paper presents the implementation of a software application and its respective results in analyzing real-time telecommunication alarm logs. The software application was developed in OMNeT++ (Objective Modular Network Testbed in C++) simulation environment using ACE (Adaptive Communication Environment) toolkit. Different working scenarios are presented in order to simulate extensions of the frequent pattern recognition algorithm: the introduction of time-constraints between alarms and the construction of a Petri net whose transitions are labeled by recognized frequent patterns of alarms.

Keywords: pattern recognition, OMNeT++ simulation environment, ACE toolkit.

I. INTRODUCTION

The volume of information transported by telecommunication networks increases and also the number of specific alarms in telecommunication networks increases. Therefore it became necessary to study different alarm correlation techniques in order to guarantee that all alarms are treated accordingly to the telecommunication networks supervision policy [3]. One of the alarm correlation techniques is to use data-mining into alarm logs to search for possible patterns (chronological sequences of alarms, also called chronicles) that repeat themselves with a certain frequency and therefore may indicate a correlation between the respective alarms. Frequent pattern recognition (chronicle recognition [4]) is used to determine possible alarm correlations but does not determine the relevance of these alarms. It is in the scope of work of the network operator or of the expert-system for network supervision to further analyze alarm correlations and to establish relevance for the recognized patterns.

For the purpose of analyzing real-time alarm logs, such as telecommunication alarm logs, we developed the theoretical aspects for a frequent pattern recognition algorithm, presented in a previous paper [8], and now we present the practical aspects following a software application that implements the given algorithm and its extensions. We also present in this paper different working simulation scenarios that were used for the purpose of assessing some performance aspects of the algorithm [7] and of its extensions by the introduction of time-constraints [2], and Petri net analysis [1].

To implement the software application we used the *OMNeT++* (Objective Modular Network Testbed in C++) simulation environment [10], previously presented in paper [9]. For the real-time communication modules we used *ACE* (Adaptive Communication Environment) toolkit [5], [6], [11]. *ACE* is an open-source software of approximately 135.000 SLOC (Source Lines Of Code).

II. SIMULATION ENVIRONMENT

In the field of telecommunication network analysis there are different simulation environments with specific facilities for addressing different simulation needs. For example, commercial simulation environments such as *COMNET*, *OPNET*, *Hyperformix Workbench*, *Mesquite CSIM* and *Simscript* address industrial simulation needs, while academic simulation environments such as *Smurph*, *NetSim++*, *OMNeT++* address laboratory and non-commercial needs. For our analysis we have chosen the open-source distribution of *OMNeT++* (latest binary *3.2p1* released on January 2006), which is well supported and documented on the respective community web site [10]. We mention though that since last year *OMNeT++* community offers also a commercial version (called *OMNEST*) which addresses industrial simulation needs. *OMNeT++* is a simulation environment based on object-oriented technology and adapted for discrete event systems.

The main advantages of *OMNeT++* are the following:

- It is not necessary to study new specific programming languages for simulation, since it integrates C++ programming code,
- It offers a complete *GUI* (Graphical User Interface) to implement and supervise processes and verify software functionality,
- Simulation is platform-independent and portable on various operating systems, including win32-based and unix-based distributions,

[1] Alcatel Romania, IT S&D Department, 9 Gh.Lazăr, 300081, Timişoara, petru.serafin@alcatel.ro
[2] "Politehnica" University of Timişoara, 2 V.Pârvan, 300223, Timişoara, alimpie.ignea@etc.upt.ro

- Structures can be quickly modified using multiple parameterization facilities, without code impact,
- Predefined classes and libraries are under continuous development and improvement in open-source software development.

Examples of simulations already implemented in *OMNeT++* include queuing systems, communication protocols and other discrete event dynamic systems simulations (*INET Framework*, *Mobility Framework*, *IPv6Suite* etc.).

OMNeT++ offers a modular architecture where components are developed in C++ programming language and then assembled into higher level components using *NED* (Network Description Language). *NED* is implemented as part of the simulation environment and contains many programming facilities and graphical definitions for implementing network topology and parameterization of processes.

The main components of *OMNeT++* are the following:

- Central simulation library,
- *NED* language compiler (*nedc*),
- *GUI* for network topology (*GNED*),
- Simulation interface (*Tkenv*),
- Command-line interface for simulation execution (*Cmdenv*),
- Graphical application for simulation results (*Plove*),
- Supporting toolkits for simulation development.

The modules can be dynamically modified during a simulation in order to take into consideration the evolution of the network topology.

The modules can have an arbitrary number of connections that are developed based on input-output ports. The usage of input-output ports allows further reusability of modules in more complex connections.

The input ports detect the presence of messages and following the validation of some execution conditions other messages are presented at the output ports.

In our simulation, messages that are transported in the network are in fact alarms or alarm patterns that will be transiting the application as tokens.

III. APPLICATION DESIGN

The general architecture of the software application for analyzing telecommunication alarm logs consists of the following specialized modules:

- *Collector* module – with the purpose of reading alarm logs using a specific collector interface with the network elements,
- *Pattern Recognition* module – with the purpose of generating candidate patterns, calculating pattern frequency and retaining frequent patterns,
- *Pattern Analysis* module – to analyze collected and generated data in order to consolidate results.

Frequent patterns of alarms that are discovered in the recognition process are presented individually to the operator to further analyze alarm correlation.

Fig. 1 presents the general architecture of the software application for frequent pattern recognition:

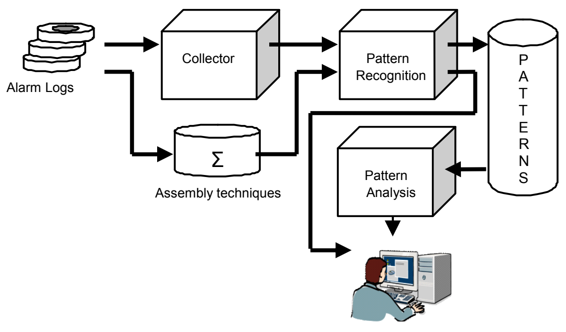

Fig. 1. General architecture for frequent pattern recognition

The detailed architecture of the software application contains the functional components and sub-modules. Our functional implementation of the pattern recognition process is presented in *Fig. 2*:

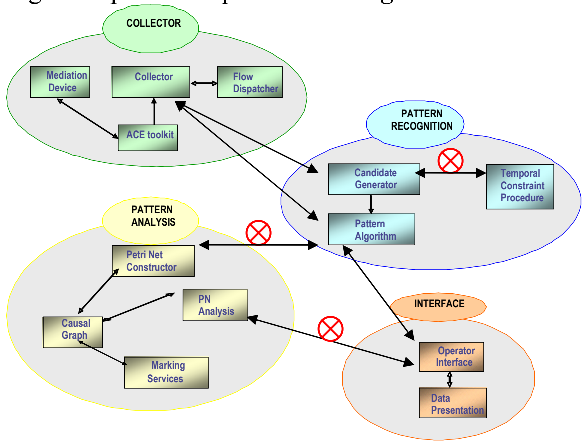

Fig. 2. Detailed architecture for frequent pattern recognition

Breakpoints are represented in *Fig. 2* by the means of the ⊗ symbol and will be used in determining the different simulation scenarios, which will be detailed in the next paragraph.

A brief description of the modules and their functionality is necessary in order to understand the simulation scenarios we will later use.

The *Collector* module is located at the entry point of the application and has the main role of mediation and flow dispatcher between the network elements and the *Pattern Recognition* module. The software implementation of the *Collector* module can be distributed in alarm concentrator units or centralized in a network supervision unit. In our *OMNeT++* implementation we centralized alarm collection in a central unit. The internal architecture of the *Collector* module uses the *ACE* toolkit as a library of platform-independent adaptive communication functions. *ACE* toolkit has the advantage of being portable on different operating systems, contributing to the overall portability of the software application. Different network supervision systems are based on different operating systems and therefore using a common library is important for reusability aspects.

Alarm data is collected by pull transfer mode which is a synchronized extraction of data block piloted by the *Collector* module. Generally the communication protocol between network elements and *Collector* module is constructor dependent. Our simulation uses *FTP* (File Transfer Protocol) to retrieve buffered alarm logs. Flow dispatcher further adapts and negotiates alarm blocks transfer through the upper level of the application. Alarm messages are transmitted and consumed by the software modules under the form of tokens.

Pattern Recognition module realizes the algorithm in its initial description: based on some assembly techniques it generates candidate patterns and then applies a formula for frequency calculation and retain only the frequent patterns to be presented to the operator and/or to the *Pattern Analysis* module.

Different assembly techniques may be used to determine patterns, depending on the prerequisite relations between alarms.

Serial assembly may be used if there is no ordering between alarms, neither by priority nor by chronology. This generates sequences of unordered alarms. For example, a sequence of alarms *(a,b,c)* serial assembled with a repeating alarm *b* results in the sequence *(a,b,c,b)*.

Parallel assembly takes into consideration a certain priority between alarms, dictated by network supervision policy. This generates sequences of ordered alarms. For example, a sequence of alarms *(a,b,c)* parallel assembled with a repeating alarm *b* results in the sequence *(a,b,b,c)*. This presumes that network supervision policy considered that *a* alarm has priority over *b* alarm, and *b* alarm has priority over *c* alarm.

Once the candidate patterns are generated, the frequent pattern recognition algorithm calculates the occurrence frequency of the candidate pattern using expression (1):

$$f_{\min}(p) = \left[\min_{p \in L} \left(\frac{a_i \in L}{a_i \in p} \right) \right] \qquad (1)$$

where *p* is the candidate pattern (included in the *L* alarm log) and a_i is the generic term for alarm occurrences included in this pattern (*i* being the alarm index in the pattern).

The algorithm then selects and retains only frequent patterns ($f \geq f_{\min}$) to be further analyzed.

To explain the pattern recognition algorithm, we consider the alarm log given by expression (2):

$$L(a,b,c) = \left\{ aca\overset{b}{\underset{c}{}}abcc \right\} \qquad (2)$$

This considered alarm log *L(a,b,c)* contains occurrences of alarms *a*, *b* and *c* (observe that at a certain time alarms *b* and *c* occur simultaneously, which is represented by a superposition of those alarms).

At each step, the algorithm generates candidate patterns of superior order, starting from the elementary order (see *Fig. 3* and *Fig. 4*). Then there is a frequency calculation based on expression (1). To explain the expression (1), we may calculate the occurrence frequency for pattern *(a,c,c)* in the alarm log *L* given by expression (2):

$$f_{\min}(a,c,c) = \left[\min_{a,c \in L} \left(\frac{a \in L}{1}, \frac{c \in L}{2} \right) \right] = 2 \qquad (3)$$

Serial assembly over *L* alarm log with a given minimal frequency $f_{\min}=2$ has the following results, represented in *Fig. 3*:

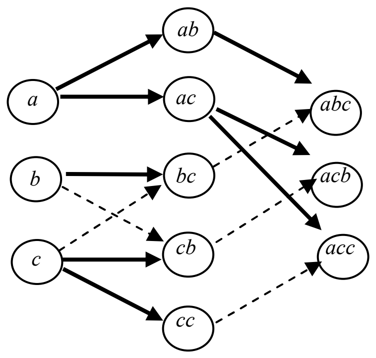

Fig. 3. Frequent patterns recognized in serial assembly

Given the same alarm log and frequency, parallel assembly results in the following frequent patterns, represented in *Fig. 4*:

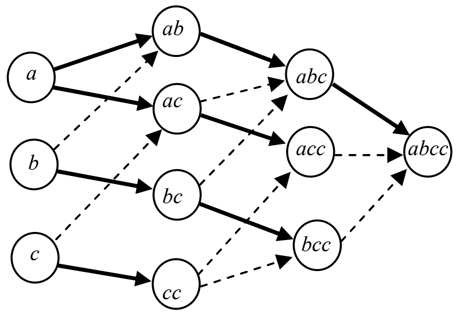

Fig. 4. Frequent patterns recognized in parallel assembly

The results of the frequent pattern recognition algorithm presented previously in *Fig. 3* and *Fig. 4* are based on simple hypothesis regarding assembly techniques and do not take into consideration temporal constraints between alarms. Therefore, the first major extension of the algorithm consists in defining and using temporal constraints in the pattern recognition process.

Temporal constraints between alarms are introduces by the following notions: given an alarm log that contains alarms *a* and *b*, the temporal constraint

between a and b is the superior limit of the temporal distance between a and b with respect to their occurrences in the pattern. For example, if we consider the occurrence $[(a,t_a)(b,t_b)]$ the temporal distance would be $T(a,b)=t_b-t_a$.
We define a temporal constraint parameter $c(T)$ that is the ratio between a and b occurrences that verify expression T over the total number of a and b occurrences in the pattern:

$$c(T)=\frac{\left|\left\{\left[(a,t_a)(b,t_b)\right]\in p_{ab}, t_b-t_a=T\right\}\right|}{\left|p_{ab}\right|} \qquad (4)$$

Following the introduction of the temporal constraints and the parameter in expression (4), we construct the temporal constraint procedure as follows:

```
Procedure Tconstraint $(a,b,c_{\min}(T))$;
/* Temporal constraint calculations */
Input
    $p_{ab}=\left\{\left[(a,t_a^i)(b,t_b^i), i=1..n\right]\right\}$ /* pattern */
    $c_{\min}(T)$ /* given minimal constraint */
Output
    $C(T)$ /* constr. set verifying $c_{\min}(T)$ */
{
/* Initialize constraint set */
    $C(T)=NULL;$
/* Initialize constraint set space */
    $S=\left\{t_b^i-t_a^i \middle| i=1..n\right\};$
/* Calculate temporal constant $k$ */
    $k=\left[c_{\min}(T)\cdot n\right];$
/* Sorting temporal space */
    Sorting $S=\{x_1\le\ldots\le x_n\};$
/* Composing and verifying $c_{\min}(T)$ */
    For $i=1; i\le n-k+1; i++$
        For $j=i+k-1; j\le n; j++$
            $C(T)=C(T)\cup\left\{\left|x_i,x_j\right|\right\};$
/* Return constraint set */
    Return $C(T)$;
}
```

The introduction of the temporal constraint procedure in the *Pattern Recognition* module further refines the recognized patterns. A situation where temporal constrains show their necessity is if the alarm log contains occurrences of a pattern in a relatively closed time frame and then also contains the same pattern detected with a very large time frame. To speed up the algorithm we define a minimal time frame during which patterns may be recognized and so we will not need to memorize a pattern once it was already recognized. This scenario will constitute one of the performance tests of the algorithm; detailed results are presented in the following paragraph.
Once frequent patterns are recognized we want to perform a first analysis of these patterns, related essentially to finding consequent patterns or patterns that include each other. One of the possible approaches for this analysis is the construction of a Petri net which transitions are labeled with the previously recognized frequent patterns, and then we want to analyze the marking situations in this Petri net. Mixing pattern recognition with Petri net assembly is a first step toward *Pattern Analysis* and it provides important information about the recognized patterns.
One of the main advantages of the *Pattern Analysis* module is that it operates almost independently from the *Pattern Recognition* module. Almost independently because it takes inputs from the recognition algorithm during the assembly of the Petri net and then helps operate on the recognition algorithm. Petri net simply provides results of the eventually consequent patterns and therefore simplifies some calculations of higher level candidate patterns during the algorithm. The theoretical bases of mixing Petri net assembly and pattern recognition are detailed in [7].

IV. PERFORMANCE RESULTS

With the previous considerations, we constitute a list of scenarios activating or deactivating functional sub-modules of the software application.
The first scenario (further referred as *Scenario 1*), consists of a simple execution of the pattern recognition algorithm, without temporal constraint procedure and without activating *Pattern Analysis* module. This provides primary results that can be compared with next scenarios.
The second scenario we use (further referred as *Scenario 2*), consist of the activation of temporal constraint procedure during the pattern recognition algorithm. Referring to *Fig. 2*, *Scenario 2* is obtained by activating the $\otimes$ symbol between the *Candidate Generator* procedure and the *Temporal Constraint* procedure. As we expected, the introduction of the algorithm does filtrate some patterns that are recognized rather late with respect to a given time frame. This leads to better performance of the overall software application.
The third scenario (noted *Scenario 3*), activates the independent module of Petri net assembly and analyses possible inconsistencies between the recognized frequent patterns. Therefore some frequent patterns will not be presented to the operator since they are included in other frequent patterns. Referring to *Fig. 2*, *Scenario 3* is obtained by activating the two $\otimes$ symbols that connect the *Pattern Analysis* module to the software application. As expected, this scenario leads to better performance of the pattern recognition.

All scenarios were simulated over the same input alarm log, in order to preserve the possible comparative arguments between the scenarios.
Considering a recorder telecommunication alarm log of 3000 occurrences of 25 types of alarms, we start by executing simulations at given minimal frequencies. For example, we chose 25, 50, 100, 250 and 500 as minimal frequencies for our calculations.
For each considered frequency we then execute the simulation and memorize or calculate following data:

- Frequent alarms,
- Generated candidate patterns,
- Frequent patterns,
- Simulation execution time.

For example, *Table 1* contains results for the simulation execution of *Scenario 1*:

Table 1

Patterns Frequency	*Frequent Alarms*	*Candidate Patterns*	*Frequent Patterns*
25	24	5817	366
50	17	4905	108
100	9	2892	28
250	3	838	11
500	1	78	0

As expected, by increasing the frequency we obtain less frequent patterns and frequent patterns results are refined by the simulation scenarios.
The synthesis graph showing frequent patterns evolution in relation to given minimal frequencies is presented in *Fig. 5*:

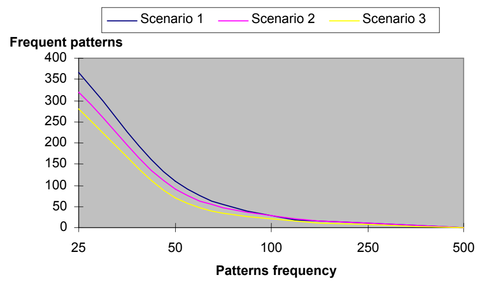

Fig. 5. Frequent patterns in simulation scenarios

Simulation time decreases by increasing minimal patterns frequency, which is explained by the fact that fewer candidate patterns are generated and calculated as they do not verify minimal frequency condition. At the extreme cases, if the desired patterns frequency is too high then the algorithm may stop at the first step of calculating single alarm frequencies. Vice-versa, by selecting a low frequency more and more candidate patterns verifies the minimum frequency and therefore the calculations become time-consuming and the simulation time increases.
Concerning the simulation scenarios' execution time, we collected the following data that is presented in *Table 2*:

Table 2

Patterns Frequency	*Scenario 1*	*Scenario 2*	*Scenario 3*
25	13:27	12:51	11:34
50	03:10	02:50	02:29
100	00:44	00:42	00:35
250	00:28	00:25	00:22
500	00:04	00:04	00:03

The synthesis graph showing simulation scenarios execution time in relation to given minimal frequencies are presented in *Fig. 6*:

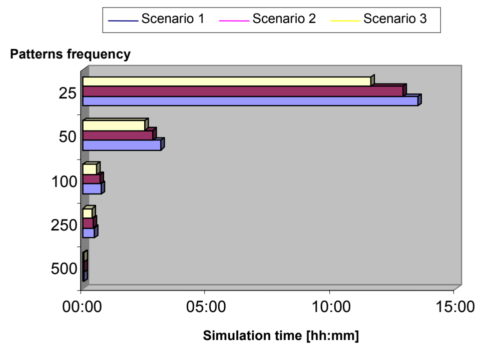

Fig. 6. Simulation scenarios execution time

Some comments of the performance results are necessary before concluding on the overall software application. First we notice that the implementation of the algorithm's extensions is improving the simulation time and also improving the quality of the solution (less frequent patterns are recognized but the relative relevance of these frequent patterns is greater, either because a time frame for the recognition process was defined or because inconsistent patterns were eliminated from the final solution).
Performance improvements presented above demonstrate the refinement of the final solution by the means of eliminating some intermediate solutions to reach a better final solution to be presented to the network operator.
Further simulations on different alarms logs produced equivalent results, depending on the topology of the alarm logs. For example, the recognition algorithm produces faster final solutions when applied over a simpler alarm log with fewer occurrences of alarms. On the contrary, when applied over a more complex alarm log that contains more occurrences of alarms, the recognition algorithm takes longer to produce both intermediate (candidate patterns) and final (frequent patterns) solutions.
Another aspect of these performance results is that it proves that the recognition algorithm itself can be extended with the help of theoretical contributions and mixing other data analysis techniques to the recognition process. Theoretical improvements include the consideration of a certain time frame limitation when recognizing patterns. This is expressed mathematically by the introduction of time

constraints between alarms and the physical application of these constraints is to filter out late occurrences of alarms in the considered alarm log.
Data analysis techniques that may help obtain a better final solution include the Petri net assembly. As demonstrated in our software application implementation, a dedicated module that constructs Petri net and then provides a short analysis of the resulting topologies increases the performance of the overall application.
It is important to mention that only some preliminary analysis was done with the help of the Petri net assembly, only for the purpose of demonstrating the possible application of this method for the scope of pattern recognition. Based on this support we may consider other methods for the scope of obtaining a better final solution of the presented algorithm.

V. CONCLUSIONS

The main outcome of the software application developed for frequent pattern recognition in alarm logs is that it proved a feasible implementation of the theoretical aspects of the recognition algorithm and some of its extensions.
Using generic project management techniques we developed the software application to support various possible simulation scenarios for the purpose of demonstrating value-added possible extensions of the recognition algorithm.
Beside the performance aspects presented in the previous paragraph, there are some interesting results about the frequent patterns themselves. For example, in a real-time situation analysis, we managed to detect a pattern that was not taken into consideration by the network operators, since it was collateral to the telecommunication network supervision policy: it was detected that an auxiliary power supply interruption caused a sequence of alarms, starting from an over-heating alarm and leading to a pattern of telecommunication-related alarms. The explanation was simple: the auxiliary power supply connected the cooling system and therefore its interruption caused cooling system malfunction and finally lead to telecommunication equipment alarms. Generally this kind of sequences of alarms demonstrates the interest in pattern recognition for the telecommunication alarm logs: it proves that some of the recognized pattern may be useful in network maintenance and supervision.
The most important factor in the analysis of frequent patterns is to focus on the initial alarm in the sequence of alarms. In most cases, the initial alarm represents the primary cause of the defect that is being signalized to the network operator. However, in real-time telecommunication systems, alarms do not always appear to the supervision network in the order in which they were produces in the network. This is caused mainly by alarm propagation delays that occur in telecommunication networks.
One of the possible solutions to the problem of considering propagation delays is to register original occurrence time in the alarm logs (and to sort the alarm log in the chronological order of appearance) or to accept larger time slots which will induce the possibility of alarms that appear to be simultaneous in the mathematical representation prior to the application of the pattern recognition algorithm.
Experts in telecommunication network supervision systems that consulted our software application concluded that frequent pattern recognition is useful in networks supervision and has potential towards further development of expert-systems applied to this field of expertise. Also, it was observed that the pattern recognition algorithm and its proposed extensions are theoretically applicable to other fields of expertise such as electrical energy network supervision or other event correlation systems analysis.

REFERENCES

[1] A. Aghasaryan, C. Dousson, "Mixing Chronicle and Petri Net Approaches in Evolution Monitoring Problems", *Proceeedings of the 12th WPD (Workshop Principles of Diagnosis)*, pp.1-7, San Sicario, March 2001
[2] F. Fessant, C. Dousson, F. Clérot, "Mining on a telecommunication alarm log to improve discovery of frequent patterns", *Industrial Conference on Data Mining (ICDM)*, Leipzig, July 2003
[3] G. Fiche, G. Hébuterne, "Trafic et performances des réseaux de télécoms", *Ed.Hermes-Science, Groupe des Ecoles de Telecommunication & Lavoisier,* Paris, 2003
[4] B. Guerraz, C. Dousson, "Chronicle Construction Starting from the Fault Model of the System to Diagnose", *International Workshop on Principles of Diagnosis,* pp.51-56, Carcassonne, 2004
[5] S. Hudson, J. Johnson, U. Syyid, "The ACE Progammer's Guide", *Ed.Addison-Westley*, October 2003
[6] D. Schmidt, S. Hudson, "C++ Network Programming : Mastering Complexity with ACE and Patterns", Volume 1, *C++ In-Depth Series, Bjarne Stroustrup, Ed. Addison-Westley*, December 2001
[7] P. Serafin, "Contribuții la analiza alarmelor în rețelele de telecomunicații", *Ph.D. Thesis, "Politehnica" University of Timişoara,* pp.91-134, December 9, 2005
[8] P. Serafin, "Algorithm for Frequent Pattern Recognition in Telecommunication Alarm Logs", *Scientific Bulletin "Politehnica" University of Timişoara, Transactions on Electronics and Communications*, Tom 50 (64), Fascicola 1, pp.30-33, September 22, 2005
[9] P. Serafin, "Network Simulation Using OMNeT++ Environment", *Scientific Bulletin "Politehnica" University of Timişoara, Transactions on Electronics and Communications*, Tom 49 (63), Fascicola 1, pp.407-411, Symposium of Electronics and Telecommunications, Timişoara, October 22-23, 2004
[10] http://www.omnetpp.org, OMNeT++ (Objective Modular Network Testbed in C++) community web site
[11] http://www.riverace.com, ACE (Adaptive Communication Environment) web site