UNIVERSITATEA "POLITEHNICA" DIN TIMIŞOARA
FACULTATEA DE ELECTRONICĂ ŞI TELECOMUNICAŢII

Ing. Petru Lucian SERAFIN

CONTRIBUŢII LA ANALIZA ALARMELOR ÎN REŢELELE DE TELECOMUNICAŢII

~ Teză de doctorat ~

Conducător ştiinţific
Prof.Dr.Ing. Alimpie IGNEA

TIMIŞOARA
2005

CUPRINS

Lista de figuri

Lista de tabele

i. Abstract

In this Doctoral Thesis I propose some contributions for alarm analysis in telecommunication networks, in order to improve the chronicle recognition algorithm to help in the process of network supervision. I introduce an evolution of the chronicle recognition algorithm by using Petri Net assembly to analyze dependencies between frequent chronicles. First, chronicle recognition algorithm is used to determine relevant frequent chronicles, which appear in the alarm log. Then a Petri Net whose transitions are labelled with the solutions of the algorithm is assembled and further analyzed in order to describe the behaviour of the telecommunication network. By mixing the advantages of Petri Net and chronicle recognition algorithm the main outcome is obtaining a chronicle-labelled Petri Net that is useful in determining the minimal set of solutions, without loosing information.

The actual phase of telecommunication services development implies that networks are increasing both in size and in complexity. Therefore, the telecommunication network monitoring system needs to handle an increasing volume of information about network events. Most of the information consists of alarms that are produced by network elements in spontaneous mode. For example, some alarms may concern functional problems of a physical element, some other alarms may indicate a software problem of the communication call control. These alarms produce an alarm flow that is gathered in alarm logs in order to be processed on-line or off-line by the network monitoring system. We need to be able to effectively manage the telecommunication network, and therefore the monitoring system must analyze the alarm logs in the search for major events in the system and eventual time correlations between alarms.

Because of the propagation delays that generally appear in telecommunication networks, the monitoring process is often mislead and therefore a complete modelling must consider time constraints. The domain of research having as purpose the analysis of telecommunication networks continues to be studied by different authors with different methods that propose more or less complex modelling. For example, some methods propose an efficient rule-based language that was especially developed to allow a complete description of network events. Some other methods introduce object-oriented programming and data mining. I propose the use of a Petri Net assembly module with the purpose of analyzing dependencies between frequent chronicles.

Starting from the algorithm of chronicle recognition, the main objective of my research was to develop the algorithm make a software project to support it with experimental results. As defined in previous papers, alarm chronicles are successions of alarms that are correlated by order of appearance and time constraint. By choosing a minimal frequency of apparition, the chronicle recognition algorithm identifies patterns

that are frequent and may be relevant to the network monitoring. The chronicle recognition algorithm does not consider isolated alarms, which should be treated separately as non-frequent events. It is important to notice that the recognized chronicles are not classified into relevant or irrelevant ones, only the network monitoring experts may further analyze those patterns using expert-system reasoning. Therefore the algorithm does not substitute itself to expert systems, it only offers a support for network supervision. Expert reasoning can be obtained by human operator over a period of time and experience with the supervised telecommunication network, or by expert-system applications following logical correlation rules.

The mixed method presented here helps telecommunication networks supervisor systems to recognize frequent successions of alarms, which are called frequent chronicles. Then, by using Petri Net assembly, the method takes advantage of the powerful modelling facilities of Petri Nets. The main idea of this mixed method is that the algorithm and the Petri Net assembly should complete each other's potential to determine solutions. Hypotheses in Petri Net that meet a consecutive or concurrent situation are used to eliminate determined solutions and therefore eliminate whole branches of candidate chronicles. Consecutive situations mean that chronicles are causally related and mutually ordered and concurrent interaction means that chronicles may result in an interleaving firing. The result is a reduction of complexity of the algorithm itself, and because the Petri Net module is launches in parallel with the main module, it will not affect the execution time for determining the complete solution. In fact, execution time proved to be better when Petri Net module is introduced, because of the reduction in complexity.

Future work may consist in further developing of the algorithm to introduce predictive procedures that will allow experts to identify frequent chronicles even before all the sequences of alarms are identified in the system. Using predictive procedures the network monitoring system may ask the alarm collector to ignore the already recognized chronicles, thus gaining some immunity on alarm loss.

Telecommunication network experts that helped analyse the experimental results that I obtained during the development of a software project to support theoretical improvements of the algorithm, agreed that the proposed software project could be used successfully in network supervision and may constitute the basis of future development of expert-reasoning supervision systems for telecommunication networks and other similar network supervision processes based on alarm analysis.

ii. Motivație

Domeniul în care îmi desfășor activitatea de o lungă perioadă de timp, în cadrul departamentului de dezvoltare de proiecte informatice pentru centralele telefonice *Alcatel 1000 E10* [1], m-a adus în contact direct cu tehnicile pentru prelucrarea alarmelor și prezentarea acestora către sistemele informatice de supervizare *NMC (Network Management Center)* [2].

Problema prelucrării alarmelor pentru a realiza supervizarea rețelelor de telecomunicații a făcut obiectul mai multor studii de specialitate, printre care menționez tezele de doctorat ale lui Armen Aghasaryan [3], Renée Boubour [25], Christophe Dousson [46] și Thang Vu Dong [131]. Teza de doctorat [3] a fost elaborată de un coleg, inginer de cercetare de la *Alcatel R&I (Research & Innovation)* la centrul din Marcoussis, Franța. Teza de doctorat [46] a fost dezvoltată de un inginer de sistem de la *CNET Francetelecom (Centre National d'Etudes des Télécommunications)* din Lannion, Franța, centru care are o strânsă colaborare cu departamentele *Alcatel E10 R&D (Research & Development)* din Lannion și Timișoara.

Aceste lucrări de referință au constituit o bază de pornire pentru studiul în domeniul analizei alarmelor în scopul recunoașterii cronicilor frecvente, care se finalizează, acum, prin prezentarea unei teze de doctorat conținând o soluție originală de abordare a problemei. Soluția este de a analiza cronicile frecvente recunoscute în jurnalele de alarme de telecomunicații folosind formalismul rețelelor Petri. Soluția a fost inițial sugerată în mai multe articole recente [6], [58] și [61], care prezintă o introducere în cadrul teoretic al problemei. Contribuția mea este dezvoltarea cadrului teoretic al analizei alarmelor și realizarea unui proiect informatic, într-un mediu de programare integrat, pentru susținerea soluției prin rezultate experimentale asupra unor simulări de laborator ale unor rețele de telecomunicații. Pe parcursul elaborării tezei, am publicat mai multe articole referitoare la acest subiect. Astfel, în articolul [113] am prezentat cadrul teoretic pentru aplicarea rețelelor Petri etichetate de cronicile frecvente recunoscute în jurnalele de alarme, în articolul [114] am prezentat o dezvoltare a algoritmului de recunoaștere a cronicilor frecvente, iar în articolul [115] o scurtă prezentare a mediul de programare *OMNeT++ (Objective Modular Network Testbed in C++)* [126], folosit pentru dezvoltarea proiectului informatic. Proiectul informatic este în regim de sursă liberă și l-am publicat pe site-ul web al comunității de programare *OMNeT++* [137]. Pentru analiza rezultatelor obținute în raport cu rezultatele publice ale altor cercetări similare, am ales compararea funcționalității proiectului informatic față de rezultatele prezentate la terminarea proiectului de cercetare *MAGDA (Modelisation et Apprentissage pour une Gestion Distribuée des Alarmes)* [134], [136], proiect comun *Alcatel*,

Francetelecom şi *INRIA (Institut National de Recherche en Informatique et en Automatique).*

Aplicarea reţelelor Petri în domeniul reţelelor de telecomunicaţii nu este o abordare nouă şi nici singulară. Deja de la începutul dezvoltării tehnicilor de comutaţie digitală pentru centralele telefonice, între anii 1980-1990, în centrele satelit numerice pentru racordarea posturilor telefonice s-au implementat module informatice, având la bază modelări cu reţele Petri. Reţelele Petri sunt utilizate, în principal, datorită capacităţilor lor de a exprima în mod simplu cauzalitatea unor evenimente într-un sistem considerat. În acest sens, proprietăţile reţelelor Petri permit o foarte bună reprezentare a relaţiilor dintre alarme în reţelele de telecomunicaţii.

Pentru realizarea practică a unui proiect informatic pentru modelarea cu reţele Petri în domeniul alarmelor de telecomunicaţii, am ales mediul integrat de dezvoltare *OMNeT++*, mediu de programare *public-source* care permite integrarea de module independente, fiind disponibil în regim de licenţă publică în mediul academic. Menţionez că în mediul profesional sunt folosite medii de dezvoltare integrate mai puternice, cum ar fi *HyPerformix™ Workbench* sau *Mesquite® CSIM*, care necesită licenţe comerciale. Consider că *OMNeT++* reprezintă un suport suficient şi eficient pentru analiza în timp real a unor jurnalele de alarme simulate în condiţii de laborator, de dimensiuni acceptabile. Dacă se doreşte însă analiza în timp real a unor jurnale de alarme de dimensiuni mari atunci mediul de dezvoltare ajunge să fie limitat în reactivitate şi nu mai este eficient pentru prelucrarea alarmelor. Rezultatele experimentale prezentate în teză sunt obţinute prin analiza în timp real a unor jurnale de alarme recuperate de la anumite reţele de telecomunicaţii având un număr redus de echipamente, pentru a nu se ajunge la problemele de limitare datorate puterii de calcul a mediului de dezvoltare.

iii. Mulţumiri

Această teză de doctorat prezintă studiul pe care l-am efectuat în ultimii ani în cadrul departamentului *CTR (Centre Technique Roumanie)* al *Alcatel România*. Prin urmare, mulţumesc domnului Gérard Guillemot, directorul departamentului *CTR* şi domnului Gilbert Jaouen, directorul anterior al departamentului, pentru susţinerea profesională şi facilitarea obţinerii de contacte şi documentaţie tehnică necesară bunei desfăşurări a activităţii mele de cercetare.

Doresc să-mi exprim întreaga g ratitudine domnului Prof.Dr.Ing. Alimpie I gnea, conducătorul ştiinţific al tezei de doctorat, pentru încadrarea sa academică şi aportul didactic deosebit şi pentru că a reuşit să îmi menţină deschis interesul către cercetarea ştiinţifică, în toţi aceşti ani de pregătire a tezei de doctorat.

Pentru realizarea proiectului informatic, mulţumesc comunităţii de programatori *OMNeT++* pentru ajutorul oferit în problemele software specifice, atât pe forumul de dezvoltare cât şi prin publicarea continuă de noi module mai performante şi cu mai multe facilităţi de programare.

Mulţumesc, de asemenea, tuturor colegilor mei de la *Alcatel* din Timişoara, Lannion şi Orvault, care m-au ajutat cu sfaturi sau recomandări ce mi-au fost de folos pentru redactarea tezei de doctorat.

Şi, nu în ultimul rând, mulţumesc părinţilor şi soţiei mele Daniela, pentru susţinerea morală şi, mai ales, înţelegerea şi afecţiunea lor, pe parcursul anilor de pregătire a doctoratului.

În contextul actual al dezvoltării telecomunicaţiilor [35], [36], [42], [43], [88], [94], [102], [103], [118], volumul de informaţii vehiculate în reţelele de telecomunicaţii este în continuă creştere, iar constrângerile pentru prelucrarea acestor informaţii în timp real constituie o problemă importantă pentru sistemele informatice de supervizare.

Din acest motiv, se impune procesarea prealabilă în timp real a fluxului de informaţii de notificare despre starea de funcţionare a echipamentelor din reţea, în sensul filtrării acestor informaţii pentru a fi prezentat sistemelor informatice de supervizare. Problema este de un interes major pentru asigurarea funcţionării în condiţii optime a reţelelor de telecomunicaţii.

Informaţiile de notificare referitoare la apariţia unor defecţiuni sau evenimente în starea de funcţionare, la un moment dat, a echipamentelor din reţea, sunt denumite, în mod generic, *alarme*. Mulţimea de alarme care este transportată într-o reţea de comunicaţii, într-un anumit interval de timp, se constituie într-un *flux de alarme*. Fluxul de alarme este analizat de către sistemele informatice de supervizare şi se înregistrează în *jurnale de alarme*. Scopul principal al analizei jurnalelor de alarme este de a diagnostica starea de funcţionare globală a reţelei de telecomunicaţii supervizată, pentru a putea lua decizii de intervenţie sau de optimizare asupra elementelor de reţea, în caz de probleme de funcţionare.

Un flux mare de alarme neprelucrate sau brute, ar duce la supra-încărcarea sistemului informatic de supervizare. De aceea, fluxul de alarme ar trebui filtrat pentru a reţine doar acele informaţii care sunt pertinente sau relevante pentru evenimentele din reţeaua de telecomunicaţii şi, eventual, de a le grupa în diferite categorii de alarme. De asemenea, se pune problema eliminării alarmelor parazite sau superflue, pentru a nu încărca procesul de supervizare. Declararea alarmelor ca fiind fie pertinente fie parazite impune intervenţia unui expert sau a unui sistem expert în procesul de supervizare.

Un prim nivel de filtrare a alarmelor se realizează deja la nivelul echipamentelor din reţea, prin definirea anumitor priorităţi pentru alarme şi folosirea unui mecanism de regularizare bazat pe priorităţi, dar o abordare sistematică a problemei filtrării alarmelor poate fi făcută doar prin aplicarea unor metode matematice.

Analiza fluxului de alarme de telecomunicaţii constituie o problemă care poate fi studiată cu ajutorul mai multor tehnici de modelare. Una dintre aceste tehnici de modelare, studiată mai recent în raport cu analiza fluxului de alarme de telecomunicaţii, este teoria reţelelor Petri. Comparativ cu alte metode de modelare, reţele Petri prezintă

avantajul major de a avea capacitatea nativă de a reprezenta foarte bine evoluția în timp a sistemelor distribuite [99].

1.1 Supervizarea unui sistem dinamic

Supervizarea unui sistem dinamic [96] constă în urmărirea evoluției sale în timp, în scopul determinării eventualelor defecte de funcționare, pornind de la informațiile oferite de interfața sistemului.

Un sistem informatic de supervizare trebuie să conțină o componentă de colectare a informațiilor de notificare, o componentă de analiză a informațiilor şi o componentă de intervenție asupra sistemului dinamic supervizat.

Arhitectura generală a unui sistem informatic de supervizare a unui sistem dinamic este prezentată în *Figura 1.1*:

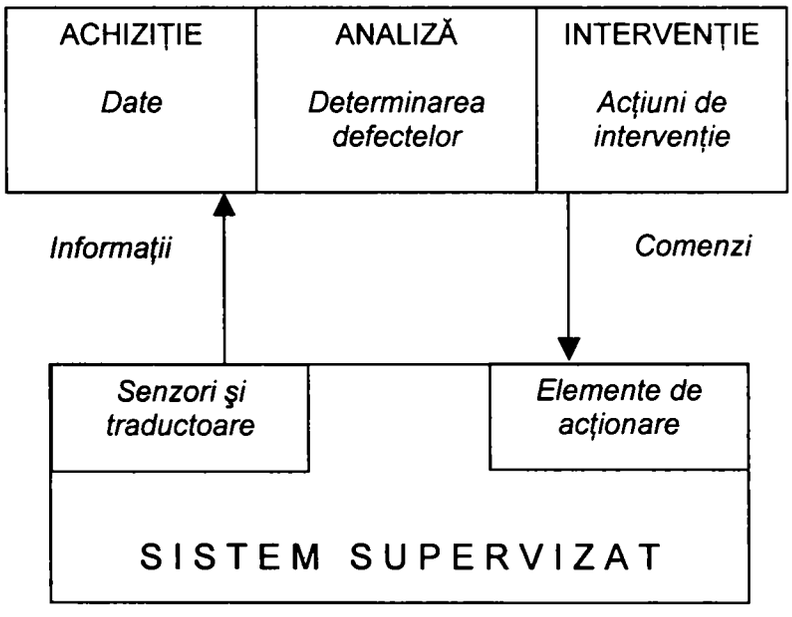


Figura 1.1 Arhitectura unui sistem informatic de supervizare : caz general

În cazul rețelelor de telecomunicații [42], [57], [90], [94], [118], aşa cum am mai arătat, informațiile de notificare privind starea de funcționare a elementelor de rețea sunt denumite, în mod generic, *alarme* (cf. definiția alarmei din paragraful §2.1) Alarmele sunt produse în mod automat şi spontan de către elementele de rețea (comutatoare, unități de racordare de echipamente telefonice, maşini logice de control al comunicațiilor telefonice, etc.) şi sunt transmise sistemului de supervizare. În prealabil, fluxul de alarme obținut este supus unui proces de filtrare [105], [107], [108], pentru a reține doar alarmele relevante din punct de vedere al funcționării elementelor de rețea care le-au emis.

Obiectivele majore al unui sistem de supervizare a unei rețele de telecomunicații sunt următoarele:

- menţinerea funcţionării în condiţii normale a reţelei,
- garantarea calităţii serviciilor de telecomunicaţii,
- urmărirea siguranţei în funcţionare.

Urmărirea siguranţei în funcţionare constituie un obiectiv aparte de diagnosticare a reţelei [50], [97], [101]. Diagnosticarea trebuie să permită detectarea, cât mai devreme posibil, a eventualelor probleme de funcţionare pentru a evita degradarea calităţii serviciilor oferite de reţea. În acest sens, trebuie detectate eventualele disfuncţiuni şi cauzele primare ale acestora şi trebuie localizate componentele implicate.

Majoritatea arhitecturilor sistemelor informatice de supervizare a reţelelor de telecomunicaţii sunt arhitecturi structurale, având o organizare modulară. Aplicaţia informatică centrală în această organizare modulară trebuie să realizeze funcţiile principale de *analizare* a fluxului de alarme din reţea. Funcţiile principale necesare pentru o analiză completă sunt : *detecţia alarmelor*, *localizarea erorilor* şi *diagnosticarea alarmelor* [27]. Scopul final urmărit de aplicaţia informatică de supervizare a reţelelor de telecomunicaţii este de a menţine calitatea serviciilor telefonice urmărind anumiţi indicatori de calitate, definiţi de standardele internaţionale [30].

Arhitectura unui sistem de supervizare pentru o reţea de telecomunicaţii este prezentată în *Figura 1.2:*

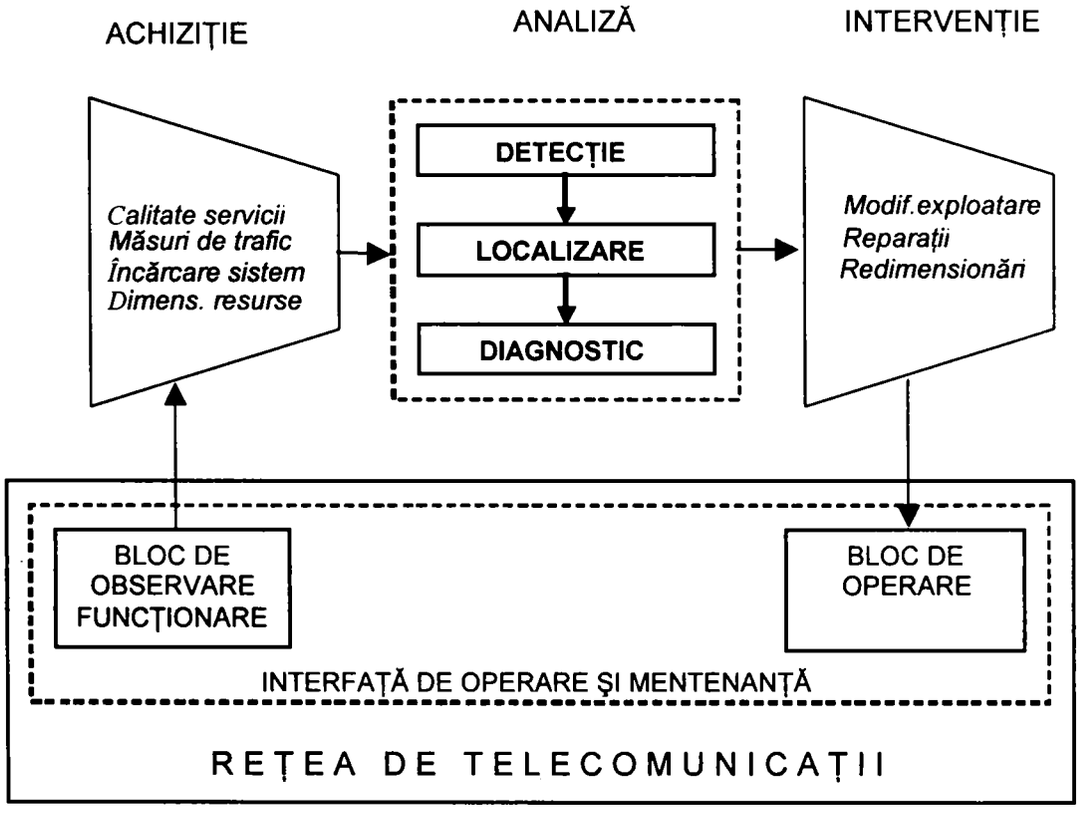


Figura 1.2 Arhitectura de supervizare : detaliere pentru reţele de telecomunicaţii

După cum se observă în figura de mai sus, procesul de achiziţie de date are următoarele obiective [42], [57], [68], [91]:

- Urmărirea calităţii serviciilor telefonice prin gestionarea alarmelor,
- Urmărirea calităţii traficului de telecomunicaţii prin măsurări de trafic,
- Supervizarea încărcării reţelei în regim de funcţionare de supra-sarcină,
- Dimensionarea resurselor pentru asigurarea serviciilor telefonice.

Nivelul calităţii serviciilor telefonice în reţea poate fi definit de un ansamblu de parametri care sunt măsuraţi, înregistraţi şi analizaţi de sistemul informatic de supervizare. Recomandarea *ITU E.420* [30] defineşte următorii 5 parametri fundamentali ai calităţii serviciilor telefonice:

- Garantarea posibilităţii clientului de a utiliza serviciile telefonice dorite,
- Oferirea unui nivel de servicii pentru iniţializarea apelului telefonic,
- Oferirea unui nivel de servicii pe durata convorbirii telefonice,
- Garantarea calităţii tehnice a convorbirii telefonice,
- Exactitatea pentru taxare şi factură telefonică.

Comportamentul acestor parametri poate fi controlat folosind anumiţi indicatori ai calităţii serviciilor [57], cum sunt, de exemplu, rata de eficacitate a apelurilor, rata de întreruperi de comunicaţie, regulile de schimbare a tarifelor, etc.

Atunci când se constată o degradare a indicatorilor de supervizare a calităţii serviciilor telefonice, trebuie colectate măsuri detaliate pentru a analiza degradarea în vederea localizării cauzei care este la originea problemelor întâlnite. Acesta este procesul principal al sistemelor informatice de supervizare, numit proces de analiză.

Procesul de intervenţie implementează deciziile luate în urma procesului de analiză şi poate include măsuri de redimensionare a resurselor reţelei, modificări de exploatare şi chiar reparaţii din punct de vedere software sau hardware asupra elementelor de reţea.

Interfaţa care permite atât achiziţia de date cât şi intervenţiile în reţea este numită Interfaţă de Operare şi Mentenanţă şi funcţionează, în general, prin comenzi de operare om-maşină. În cazul centralei telefonice *Alcatel 1000 E10* [95] aceste comenzi sunt denumite *RHM (Relations Homme-Machine)*, iar rezultatele acestor comenzi precum şi rezultatele furnizate de centrala telefonică sunt prezentate operatorului pe diferite console de vizualizare, numite *LFN (Logical File Name)*. Pentru vizualizarea alarmelor în cazul centralei *E10* sunt definite 3 console de vizualizare *LFN* corespunzătoare celor 3 categorii de alarme pe care le voi detalia în paragraful §2.1. Această soluţie de vizualizare a alarmelor pe *LFN* a fost necesară pentru a permite operatorului din centrul local de supervizare a funcţionării centralei telefonice *E10* să poată urmări anumite categoriile de alarme şi pentru a separa astfel vizualizările acestor categorii de alarme faţă de alte

vizualizări ale altor procese de observaţie sau de operare care se realizează asupra centralei.

Vizualizarea alarmelor pe *LFN* este o soluţie de prezentare adecvată atunci când numărul de alarme prezentat operatorului este relativ redus pentru a-i permite acestuia să intervină în scopul reparării problemei care a generat alarmele. *LFN*-urile de alarme reprezintă deci un prim nivel de filtrare a alarmelor, pe baza categoriilor lor de prelucrare (intervenţie imediată, intervenţie decalată sau fără imperativ de intervenţie). Dar această filtrare pe criterii predefinite nu oferă întotdeauna posibilitatea de a determina în ansamblu toate alarmele relative la un anumit proces sau echipament.

Alarmele sunt exprimate sub diferite forme, cel mai des se regăsesc sub forma unei succesiuni de caractere alfanumerice, având o anumită semnificaţie. De exemplu, alarmele se pot regăsi sub forma unui text explicit sau o serie de caractere alfanumerice care codifică anumite cauze de eroare relative la un anumit proces sau echipament. În general, alarmele se regăsesc sub forma unor codificări care respectă anumite reguli de denumire. Respectarea acestor reguli de denumire ale alarmelor permite regruparea semnificaţiilor alarmelor într-un document centralizator sub forma unui dicţionar de alarme [2], [95].

În plus de vizualizarea locală a alarmelor, centrala telefonică permite transmiterea fluxului de alarme pentru vizualizarea distantă de la un punct central de supervizare a reţelei. Fluxul de alarme care este transmis către sistemul de supervizare este înregistrat în log-uri sau jurnale de alarme, pentru a fi analizate.

Anumite alarme pot fi consecinţe ale altor alarme, intervenite anterior, datorită înlănţuirii de evenimente într-o reţea de telecomunicaţii. Problema care se studiază, în continuare, este de a defini o metodă de diagnosticare a evenimentelor din reţelele de telecomunicaţii pornind de la recunoaşterea unor secvenţe de alarme.

1.2 Metode de supervizare şi diagnosticare

Pentru a soluţiona problema de diagnosticare [20], [24], în cazul sistemelor cu evenimente discrete cum sunt reţelele de telecomunicaţii, în practică, se folosesc în mod frecvent două metode majore:

- Prima metodă este modelarea pe baza experienţei şi a raţionamentului unui expert în supervizarea reţelelor. Avantajul acestei metode constă în faptul că este extrem de eficientă în localizarea rapidă a cauzelor primare, dar prezintă dezavantajele unei achiziţii lente în timp a expertizei şi probleme de continuitate a expertizei, atunci când reţeaua evoluează prin integrarea de noi echipamente. De asemenea, un alt dezavantaj ar fi fenomenul de erodare a expertizei în timp, fie datorită

schimbărilor care apar în structura reţelei de telecomunicaţii, fie datorită creşterii fluxului de informaţii, după cum este prezentat în lucrarea [105].

- A doua metodă este bazată pe modelarea explicită a stărilor de funcţionare şi de disfuncţionare. Avantajul ar fi că se separă cunoştinţele despre funcţionarea reţelei faţă de raţionamentul propriu-zis de diagnosticare, obţinându-se astfel modele generice şi adaptabile [47]. Dezavantajul major este că această metodă necesită o modelare complexă, necesitând medii evoluate de modelare. Menţionăm că în cazul reţelelor de telecomunicaţii anumite disfuncţionări pot reprezenta un mod acceptabil de lucru, cum ar fi de exemplu funcţionarea în regim de suprasarcină.

Mai multe lucrări din literatura de specialitate prezintă diferite abordări posibile ale problemelor de analiză şi modelare [44], [70], [89], [92], [105].

În lucrarea [105] se definesc reguli eficiente de filtrare, descrise pe cele trei nivele de operare ale reţelelor de telecomunicaţii:

- reguli de topologie, la nivelul reţelei,
- reguli de interdependenţă, la nivelul alarmelor,
- reguli expert, la nivelul sistemului de supervizare.

O dezvoltare a problemei de filtrare este descrisă în lucrarea [92], unde se prezintă un sistem de corelare a evenimentelor folosind tehnici de programare bazate pe obiecte prin construirea unor arbori de corelare a alarmelor.

Anumite tehnici specifice au fost dezvoltate pentru a lua în considerare constrângerile temporale dintre alarme, conform lucrărilor [44] şi [70]. În aceste lucrări, se foloseşte preponderent metoda de expertiză pentru a crea algoritmi de filtrare, pornind de la anumite reguli de asociere sau de grupare a alarmelor. Lucrările [119], [120], [121], [122], [123] şi [124] folosesc noţiuni de modelare calitativă.

Alte tehnici utilizează modele de comportament sub forma unor automate definite în prealabil de un sistem expert. Lucrările [4] şi [44] prezintă definirea automatelor expert pentru a construi un simulator care să genereze succesiuni de alarme etichetate prin defectele de funcţionare care le provoacă. În ipoteza unor astfel de colecţii de fenomene bine identificate şi etichetate, în lucrările [21], [22] şi [23] se propune generalizarea acestor succesiuni de alarme, determinând ansamblul lexical de atribute care sunt comune tuturor exemplelor unei defecţiuni şi care nu sunt recunoscute prin nici un contra-exemplu. Pentru această etapă de filtrare (discriminare şi generalizare), în lucrările [60] şi [63] se propune utilizarea unor tehnici de programare cu logică inductivă.

Problema exploatării informaţiilor de supervizare din reţelele de telecomunicaţii pentru determinarea frecvenţei de apariţie a unor succesiuni de alarme este studiată în lucrările [47] şi [51].

Un aspect important, care trebuie luat în considerare în domeniul reţelelor de telecomunicaţii, îl constituie absenţa alarmelor, de exemplu, atunci când un echipament

din rețea nu mai transmite informații asupra stării sale de funcționare. Acest aspect poate fi tratat sub forma reprezentării evoluției alarmelor în timp. Lucrarea [44] propune o astfel de reprezentare a evoluției alarmelor în timp, introducând noțiunea de cronică de alarmă, ca fiind o succesiune de alarme ordonate temporal.

Modelul matematic necesar pentru descrierea acestor metode nu este întotdeauna accesibil şi aplicabil rețelelor de telecomunicații, deoarece operatorii rețelelor trebuie să supervizeze, în general, echipamente de la mai mulți producători şi prin urmare, nu cu acelaşi nivel de cunoştințe despre aceste echipamente, aşa cum le cunosc producătorii acestora. Constructorii de echipamente dețin cunoaşterea precisă a comportamentului echipamentelor pe care le produc.

Menționăm aici un aspect general referitor la comunicațiile de date [88]: sistemele de supervizare pentru comunicațiile de date prezintă mai multe arhitecturi posibile pentru procesul de diagnosticare. Arhitecturile de diagnosticare [37] pot fi clasificate în următoarele categorii:

- diagnostic centralizat,
- diagnostic distribuit,
- diagnostic ierarhic,
- diagnostic ierarhic distribuit.

Aceste arhitecturi de diagnostic pot fi reprezentate ca în *Figura 1.3*:

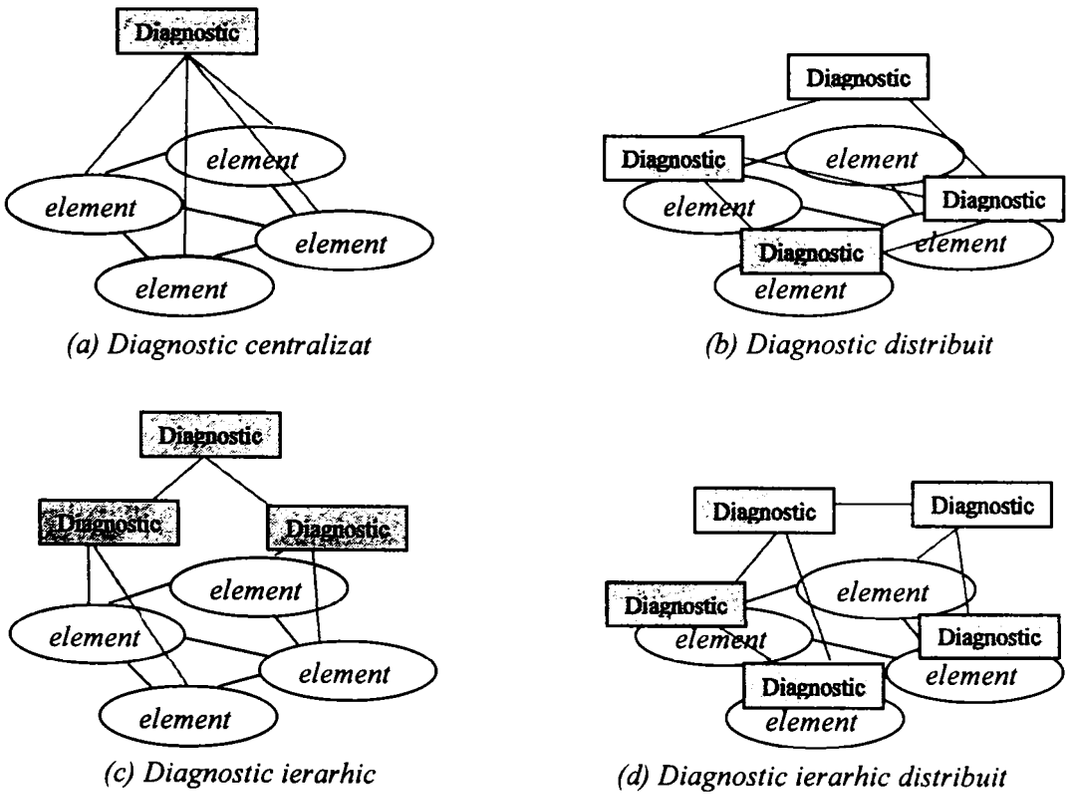


Figura 1.3 (a) Diagnostic centralizat (b) Diagnostic distribuit (c) Diagnostic ierarhic (d) Diagnostic ierarhic distribuit

Alegerea arhitecturii de diagnostic este importantă pentru a decide asupra localizării modulelor informatice de analiză a alarmelor. În cazul diagnosticului centralizat, analiza alarmelor se va face la nivelul central al sistemului de supervizare a reţelei, iar în cazul diagnosticului distribuit alarmele se vor analiza la nivelul elementelor de reţea [96], [97], [98].

1.3 Organizarea tezei

Teza este organizată în şase capitole, dintre care primele 5 capitole prezintă cadrul teoretic al problemei studiate, în toate aspectele care intervin, iar *Capitolul 6* conţine prezentarea proiectului informatic şi analiza rezultatelor experimentale.

După partea introductivă din *Capitolul 1*, în *Capitolul 2* se prezintă noţiunile teoretice referitoare la gestionarea alarmelor, prezentarea algoritmului generic de recunoaştere a cronicilor frecvente şi exemplificarea acestuia asupra unui caz concret. De asemenea, se prezintă o nouă metodă pentru extensia algoritmului prin introducerea unei proceduri pentru calculul constrângerilor temporale. Pentru dezvoltarea algoritmului prezentat în *Capitolul 2* s-a folosit teoria temporală bazată pe momente. De aceea, în *Capitolul 3* se prezintă teoriile temporale generale şi se detaliază algebra momentelor.

În *Capitolul 4* se introduc bazele teoretice pentru descrierea reţelelor Petri iar în *Capitolul 5*, noţiunile teoretice despre regulile de asociere şi câteva exemple originale de asociere a unor alarme.

Capitolul 6 începe prin descrierea mediului de programare utilizat, OMNeT++ şi prezentarea în detaliu a modulelor proiectului informatic realizat. La finalul *Capitolului 6* se prezintă rezultatele experimentale obţinute. În încheiere se prezintă concluziile şi contribuţiile originale referitoare la problema studiată.

Capitolul 2. GESTIONAREA ALARMELOR

2.1 Definiția alarmei

În *Dicționarul explicativ al limbii române – DEX* [133], se prezintă următoarea definiție pentru noțiunea de *alarmă*:

Definiția 2.1

Alarm//ă ~e *s.f.* 1) Semnal prin care se anunță apropierea unui pericol iminent; alertă; **a da** (*sau* **a suna**) **~a**. 2) Sentiment de neliniște, de îngrijorare la apropierea unei primejdii. - din Fr. *alarme*.

În rețelele de telecomunicații, toate perturbările care au consecință asupra calității serviciilor telefonice trebuie semnalate operatorului și sistemului informatic de supervizare prin mijloace vizuale și sonore corespunzătoare, denumite, în mod generic, alarme. O exprimare echivalentă a definiției alarmei, în cazul rețelelor de telecomunicații, este că aceasta reprezintă un semnal de avertizare care apare în urma detecției unei degradări a serviciilor telefonice.

Din punct de vedere *semantic*, alarmele prezintă patru caracteristici principale (indexul, momentul de apariție, tipul și categoria) și mai multe caracteristici complementare (gradul de prioritate, obiectul la care se referă, textul explicativ pentru descrierea alarmei)

Indexul de definiție al unei alarme este o referință, în general în format alfanumeric, care respectă o anumită regulă de denumire. Aceste referințe alfanumerice sunt regrupate în dicționare de alarme, indexate alfabetic, unde sunt descrise alarmele și acțiunile care trebuie efectuate de operator pentru a repara elementul fizic sau logic care a produs respectiva alarmă.

Momentul de apariție al alarmei în rețea este exprimat, în general, în formatul clasic (data și ora), dar se poate găsi, în anumite rețele telefonice, sub forme mai speciale (de exemplu numărul zilei din anul curent în loc de exprimarea clasică a datei).

Tipul de alarmă descrie originea tehnică a alarmei. Se disting două mari familii de tipuri de alarme:

- alarmele interne, generate de elementele componente ale rețelei,
- alarmele externe, generate în afara rețelei, dar care sunt transportate în rețea în regim de tranzit.

Alarmele de tip intern se clasifică în următoarele 4 sub-tipuri:

- Alarme interne de comunicaţie, sunt alarme generate în urma unei erori de funcţionare a elementelor de reţea, erori care pot provoca chiar o degradare generală a funcţionării reţelei de telecomunicaţii.
- Alarme interne de taxare, sunt alarme generate în raport cu sistemul de taxare al comunicaţiilor, fie la depăşirea unor nivele de taxare (cum sunt, de exemplu, schimbările de tarif noapte-zi), fie la o eroare în sistemul de colectare a taxelor sau a facturilor detaliate.
- Alarme interne de transmisie, sunt alarme generate de erori în sistemul de transmitere de mesaje numerice.
- Alarme interne de supraveghere reţea, sunt alarme generate în cazul detectării unor anomalii de funcţionare a reţelei, în general fiind sub forma unor depăşiri ale unor niveluri limită.

Alarmele de tip extern se clasifică în mai multe sub-tipuri, dintre care se pot enumera alarmele de întrerupere a alimentării cu energie electrică, alarmele de climatizare sau de încălzire, alarmele de efracţie etc.

Categoriile alarmelor se definesc având la bază obiective de disponibilitate a sistemului de supervizare pentru garantarea calităţii serviciilor telefonice [30]. Categoriile permit definirea timpului de intervenţie logistică şi de reparaţie, asociat unei alarme.

În mediul profesional, se definesc 3 categorii de alarme:

- Alarme cu intervenţie imediată, pentru alarmele care necesită o intervenţie de logistică şi reparaţie urgentă, cu un timp de intervenţie de maxim 4 ore,
- Alarme cu intervenţie decalată, pentru alarmele cu un grad de gravitate mai mic, dar care necesită un timp de intervenţie stabilit la 12½ ore,
- Alarmele fără imperativ de intervenţie, pentru alarmele care pot fi rezolvate într-un timp de intervenţie de 72 de ore.

După cum am mai spus, caracteristicile complementare ale alarmelor sunt gradul de prioritate, obiectul la care se referă şi textul explicativ.

Gradul de prioritate al alarmelor permite operatorului reţelei de telecomunicaţii să garanteze semnalizarea alarmelor esenţiale pentru buna funcţionare a reţelei. Alarmele mai puţin prioritare pot fi ignorate, eventual, în cazul depăşirii unui anumit nivel de umplere a zonei de memorare a alarmelor, în cazul intervenţiei unui mecanism de regularizare a alarmelor. Prioritatea poate varia pe o scară de la 0 (prioritare maximă) la n (prioritate minimă). Menţionăm că, în mediul profesional, gradul de prioritate nu este prezentată sistemului de supervizare pentru a nu destabiliza funcţionarea aplicaţiilor informatice distante. Gradul de prioritate este deci valabil doar pentru modul local de gestionare a alarmelor şi nu este retransmis către sistemul de supervizare distant.

O alarmă se referă întotdeauna la un obiect. În acest sens, noţiunea de obiect al alarmei este o extensie a noţiunii de element de reţea şi se poate referi la:

- Un element de rețea sau *stație materială* care a produs alarma,
- Un modul informatic sau *stație logică* de unde s-a produs alarma.

În general, textul explicativ pentru descrierea alarmei este un text cu informații complementare despre respectiva alarmă. Textul explicativ poate fi astfel parametrat încât să indice şi alte informații complementare, de exemplu localizarea fizică a echipamentului de rețea care a produs alarma.

2.2 Metode de analiză a alarmelor

În continuare voi prezenta o metodă de abordare a filtrării jurnalelor de alarme în rețelele de telecomunicații utilizând metoda recunoaşterii fenomenelor recurente prin modelare cu rețele Petri [113], [114]. Principiul de bază al metodei este de a analiza jurnalele de alarme pentru a determina succesiunile de alarme care se repetă cu o anumită frecvență. Alarmele respective pot fi grupate în categorii de alarme, pentru a reduce numărul de notificări prezentate sistemului de supervizare.

Rolul modulului de analiză a alarmelor este de a traduce informațiile transmise de diferitele module de detecție şi de a le grupa pentru a face un diagnostic sau pentru simpla editare a informațiilor sub o anume formă, de a edita alarmele în timp real pe fişiere logice sau console de vizualizare, de a retransmite alarmele active către centrul de supervizare, de a dispeceriza alarmele în funcție de priorități.

Metoda de analiză a jurnalelor de alarme se bazează pe trei procese elementare care sunt descrise în continuare :

- Prima componentă a analizei alarmelor este descoperirea alarmelor recurente [53], [54], [58], [114]. Această primă componentă se constituie în etapa de *detecție* a algoritmului. În etapa de detecție nu putem concluziona dacă alarma este pertinentă sau nu pentru constituirea unei cronici de alarme.
- A doua componentă a analizei este determinarea pertinenței sau a relevanței alarmelor construind grafurile de dependență. Această componentă constituie etapa de *localizare* a alarmelor în sistem.
- A treia componentă este etapa de *diagnosticare*. Problemele care apar în această etapă se datorează dimensiunilor şi complexității rețelelor, mascarea şi chiar pierderea unor alarme din cauza saturării tampoanelor de memorare, întârzierile de propagare a alarmelor în rețea şi uneori dinamica resurselor din rețea prin anumite schimbări de configurare.

Metoda de gestionare a alarmelor studiată, în continuare, se bazează pe analiza frecvenței de apariție a alarmelor în jurnalele de alarme. Se caută evidențierea unor secvențe tipar sau a unor succesiuni de alarme care se repetă mai des, numite *cronici* [6], [46], [48].

În aplicarea metodei nu va conta dacă o secvenţă tipar ce apare frecvent va permite reducerea fluxului de informaţii prezentate operatorului sau sistemului de supervizare, ci doar dacă acea cronică va corespunde unui defect de funcţionare. Această clasificare în *cronici parazite* şi *cronici relevante* trebuie însă făcută de un expert în analiza reţelei de telecomunicaţii.

În anumite cazuri, se poate ajunge la situaţii extreme când o succesiune de alarme se produce foarte rar şi poate fi pierdută în mulţimea alarmelor parazite. Aceste cazuri sunt numite *iregularităţi* ale jurnalului de alarme şi trebuie să fie luate în considerare cu aceeaşi pondere ca şi cronicile relevante.

Pentru studierea cronicilor relevante trebuie să definim termenii care intervin. Astfel, ordinea alarmelor în cadrul unei cronici este specificată de o constrângere temporală. Un exemplu de constrângere temporală, exprimată textual, este următorul: *„o alarmă se poate produce la maxim 10 secunde după o altă alarmă"*.

O abordare asemănătoare se găseşte în lucrarea [80], dar aici nu sunt tratate decât două tipuri de cronici (numite *episoade*). Episoadele pot fi *paralele* (alarme neordonate) sau *seriale* (alarme total ordonate). Singura constrângere temporală autorizată este aceea de limită maximă de timp (dimensiunea domeniului definit ca fereastră de observaţie) care trebuie definită de către utilizator. Lucrarea, având caracter teoretic, propune de asemenea, câteva extensii ale tratamentului episoadelor paralele şi seriale, dar fără a prezenta o implementare practică.

Cronicile de alarme, definite în prezenta lucrare, sunt mai complete decât cele din lucrarea [80], deoarece includ noţiuni de ordonare parţială temporală a alarmelor prin introducerea unor calcule privind constrângerile temporale dintre alarme.

2.3 Jurnale de alarme

Definiţia unui *jurnal de alarme* se poate enunţa astfel:

Definiţia 2.2

Un jurnal de alarme este o listă de alarme ordonate cronologic după momentele lor de apariţie în reţea.

Astfel, într-un jurnal de alarme ajung să se înregistreze informaţii transmise de diferitele elemente de reţea, referitoare la starea lor de funcţionare. Astfel, într-un jurnal de alarme se pot înregistra informaţii pentru o anumită perioadă de timp. Prin noţiunea de jurnal se înţelege implicit înregistrarea pentru o perioadă de o zi (fr. *jour*). În general, în reţelele de telecomunicaţii, înregistrarea alarmelor se face zilnic, iar operaţia se numeşte *jurnalizarea* alarmelor. Jurnalele pot fi regrupate, prin adăugare, pentru a obţine

înregistrări *hebdomadare* sau *lunare*. În sens invers, perioada de timp a jurnalelor poate fi descompusă în mai multe *ferestre de observaţie*.

Momentele de apariţie ale alarmelor sunt considerate în raport cu sistemul de supervizare şi nu în raport cu elementele de reţea care produc alarmele, deoarece în reţea pot să apară întârzieri în propagarea alarmelor.

Pentru a exemplifica un jurnal de alarme, în *Figura 2.1* se prezintă un extras pentru un caz real de înregistrare a funcţionării a două elemente de reţea, denumite UNIT1 şi UNIT2. Din motive de mărime a jurnalului de alarme prezentăm o înregistrare pe o fereastră de observaţie de 15 minute, suficientă pentru analiza care va fi efectuată în continuare:

```
Index / Data / Ora / Tip / Categorie / Obiect / Text explicativ
-------------------------------------------------------------
*A001/05-05-25/12H01/TYP=COM/CAT=WI/UNIT1 IN SERVICE
*A009/05-05-25/12H04/TYP=COM/CAT=IM/UNIT1 OUT OF SERVICE
*A001/05-05-25/12H05/TYP=COM/CAT=WI/UNIT1 IN SERVICE
*A007/05-05-25/12H05/TYP=COM/CAT=ID/UNIT2 OVERLOAD
*A007/05-05-25/12H07/TYP=COM/CAT=ID/UNIT1 OVERLOAD
*A009/05-05-25/12H07/TYP=COM/CAT=IM/UNIT1 OUT OF SERVICE
*A001/05-05-25/12H10/TYP=COM/CAT=WI/UNIT1 IN SERVICE
*A009/05-05-25/12H11/TYP=COM/CAT=IM/UNIT2 OUT OF SERVICE
*A007/05-05-25/12H13/TYP=COM/CAT=ID/UNIT1 OVERLOAD
*A009/05-05-25/12H14/TYP=COM/CAT=IM/UNIT1 OUT OF SERVICE
*A001/05-05-25/12H15/TYP=COM/CAT=WI/UNIT2 IN SERVICE
*A009/05-05-25/12H15/TYP=COM/CAT=IM/UNIT1 OUT OF SERVICE
```

Figura 2.1 Exemplu de jurnal de alarme

În raport cu definirea caracteristicilor alarmelor din paragraful §2.1, alarmele din acest exemplu conţin caracteristicile principale (tip şi categorie) şi alte caracteristici complementare (index de definiţie, data şi ora apariţiei în reţea, textul de descriere a alarmei). Indexul, notat **Annn* permite referirea la un dicţionar de alarme, indexat după această caracteristică. Toate alarmele din acest exemplu sunt de tip comunicaţie *TYP=COM*. Categoriile din acest exemplu respectă cele 3 categorii standardizate (*WI* – fără intervenţie, *ID* – intervenţie decalată, *IM* – intervenţie imediată). Obiectele la care se referă alarmele sunt două unităţi din reţea, iar textul explicativ este de 3 feluri.

Divizarea jurnalelor de alarme în ferestre de observaţie este necesară atunci când numărul de alarme înregistrate este foarte mare în raport cu puterea de calcul a sistemului

de supervizare. Rezultatele obținute pe timpul ferestrelor de observație pot fi regrupate în final pentru a obține rezultatele referitoare la perioada totală a jurnalului de alarme respectiv.

Fereastra de observație pentru jurnalul de alarme prezentat în *Figura 2.1* poate fi exprimată ca un ansamblu temporal *complet ordonat* de momente de apariție, notat *t=[1:15]*, având $t \in T(J)$, unde $T(J)$ este perioada globală de înregistrare a alarmelor în jurnal.

Exprimarea matematică a unei alarme este un cuplu de elemente (x^i, t_0), unde x^i este tipul alarmei, referitor la elementul de rețea i, iar t_0 este momentul de apariție al alarmei. Folosind aceste notații, din jurnalul de alarme din *Figura 2.1*, referitor la elementul de rețea UNIT1, se pot evidenția tipurile de alarme a^1 pentru IN SERVICE, b^1 pentru OVERLOAD şi c^1 pentru OUT OF SERVICE. În mod similar, referitor la elementul de rețea UNIT2se pot exprima tipurile de alarme a^2, b^2 şi c^2.

Cu aceste notații matematice ale alarmelor în jurnalul din *Figura 2.1*, se obține exprimarea din *Figura 2.2*:

Index / Data / Ora / Tip / Categorie / Obiect / Text explicativ	
*A001/05-05-25/12H01/TYP=COM/CAT=WI/UNIT1 IN SERVICE	$(a^1.1)$
*A009/05-05-25/12H04/TYP=COM/CAT=IM/UNIT1 OUT OF SERVICE	$(c^1.4)$
*A001/05-05-25/12H05/TYP=COM/CAT=WI/UNIT1 IN SERVICE	$(a^1.5)$
*A007/05-05-25/12H05/TYP=COM/CAT=ID/UNIT2 OVERLOAD	$(b^2.5)$
*A007/05-05-25/12H07/TYP=COM/CAT=ID/UNIT1 OVERLOAD	$(a^1.7)$
*A009/05-05-25/12H07/TYP=COM/CAT=IM/UNIT1 OUT OF SERVICE	$(c^1.7)$
*A001/05-05-25/12H10/TYP=COM/CAT=WI/UNIT1 IN SERVICE	$(a^1.10)$
*A009/05-05-25/12H11/TYP=COM/CAT=IM/UNIT2 OUT OF SERVICE	$(c^2.11)$
*A007/05-05-25/12H13/TYP=COM/CAT=ID/UNIT1 OVERLOAD	$(b^1.13)$
*A009/05-05-25/12H14/TYP=COM/CAT=IM/UNIT1 OUT OF SERVICE	$(c^1.14)$
*A001/05-05-25/12H15/TYP=COM/CAT=WI/UNIT2 IN SERVICE	$(a^2.15)$
*A009/05-05-25/12H15/TYP=COM/CAT=IM/UNIT1 OUT OF SERVICE	$(c^1.15)$

Figura 2.2 Reprezentarea matematică a unui jurnal de alarme

Am ales această determinare a tipurilor de alarme ținând cont de succesiunea logică a stărilor unui element de rețea. Astfel, presupunem că starea inițială de funcționare a elementelor de rețea studiate este operațională (IN SERVICE), apoi considerăm cazul unei stări de funcționare în suprasarcină (OVERLOAD) şi cazul unei stări

de nefuncționare (OUT OF SERVICE). Alegerea în ordine alfabetică a tipurilor de alarme va fi folosită ulterior pentru a facilita ordonarea alarmelor.

Facem o observație suplimentară referitor la ipotezele de lucru: presupunem că alarmele sunt în mod corect datate în jurnalul de alarme, eliminând astfel eventualele iregularități datorate întârzierilor cu care sunt notificate anumite alarme, dar vom lua în considerare eventualele întârzieri datorate propagării mesajelor în rețea atunci când vom defini algoritmul pentru sistemul de recunoaştere a cronicilor din jurnalul de alarme.

Cu notațiile introduse anterior, pentru tipurile de alarme şi momentele de apariție, jurnalul de alarme din poate fi exprimat matematic astfel:

$$J = \{(a^1,1)(c^1,4)(a^1,5)(b^2,5)(b^1,7)(c^1,7)(a^1,10)(c^2,11)(b^1,13)(c^1,14)(a^2,15)(c^1,15)\} \quad (2.1)$$

Se observă că alarmele care intervin în relația *(2.1)* sunt distincte, adică identificabile, în mod unic, prin tip de alarmă şi momentul ei de apariție. De asemenea, se observă că alarmele referitoare la primul element de rețea sunt *repetitive*, adică se regăsesc de mai multe ori în jurnal.

2.3.1 Momentele de apariție ale alarmelor

Din jurnalul exprimat prin relația *(2.1)* se pot extrage două jurnale, câte unul pentru fiecare din cele două elemente de rețea care au emis alarmele. Această extragere relativă la un anume obiect este folosită pentru a distinge elementele de rețea supervizate.

Pentru elementul de rețea UNIT1, expresia matematică a jurnalului de alarme este următoarea:

$$J^1_{t=[1..15]} = \{(a^1,1)(c^1,4)(a^1,5)(b^1,7)(c^1,7)(a^1,10)(b^1,13)(c^1,14)(c^1,15)\} \quad (2.2)$$

Pentru elementul de rețea UNIT2, jurnalul de alarme corespunzător este:

$$J^2_{t=[1..15]} = \{(b^2,5)(c^2,11)(a^2,15)\} \quad (2.3)$$

Pentru exemplificarea grafică a extragerii alarmelor referitoare numai la elementul de rețea UNIT1, se pot scoate în evidență alarmele luate în considerare, aşa cum este prezentat în *Figura 2.3*:

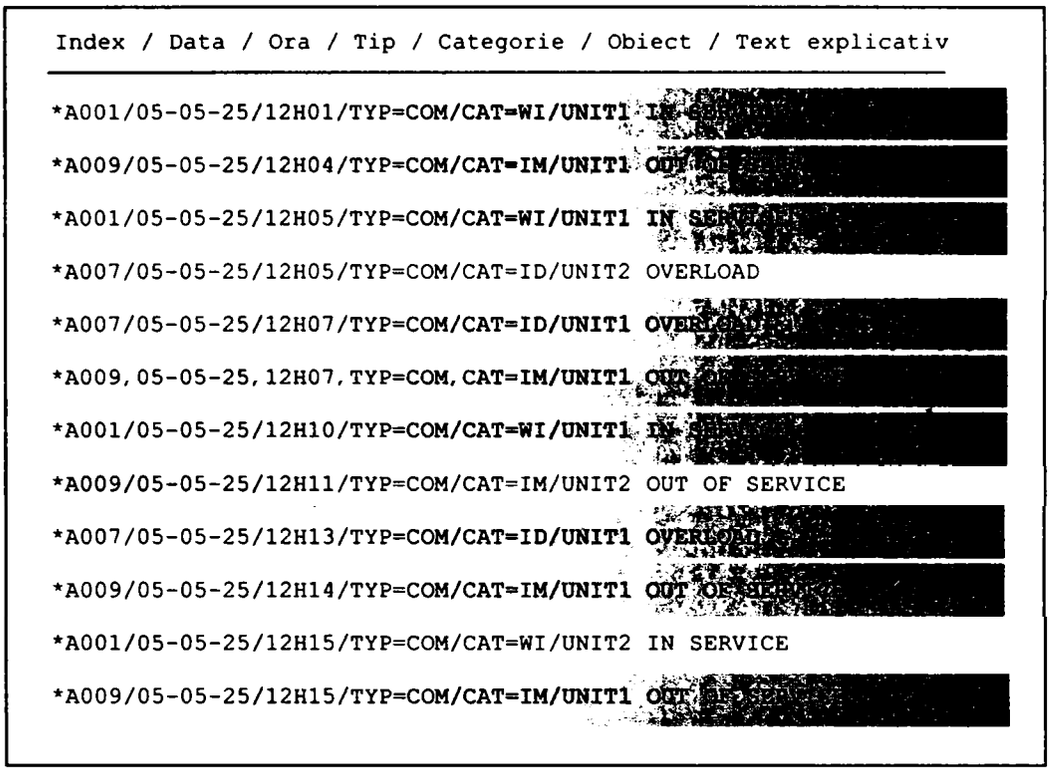


Figura 2.3 Extragerea alarmelor referitoare la un obiect

Reprezentarea grafică a jurnalului de alarme din relaţia ***(2.2)*** pentru elementul de reţea UNIT1 este următoarea:

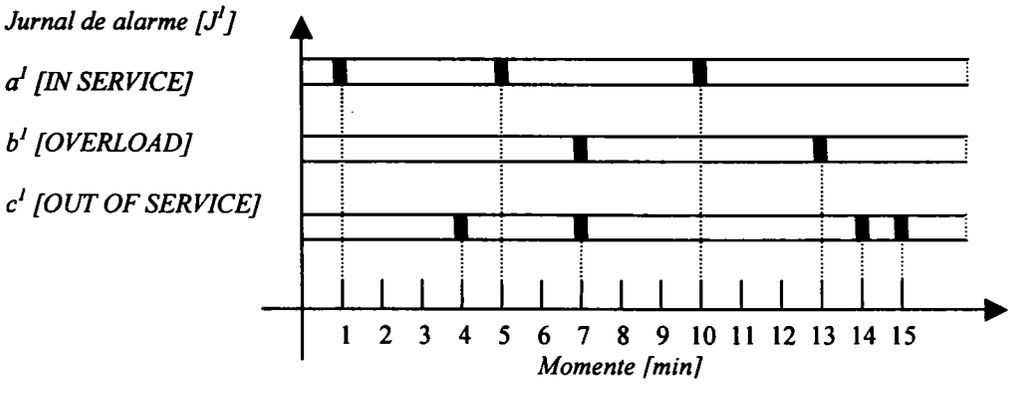


Figura 2.4 Reprezentarea grafică a momentelor alarmelor

O remarcă importantă referitoare la relaţiile ***(2.1)*** şi ***(2.2)*** este că acestea exprimă alarme care apar simultan, cum este cazul alarmelor $(a^1,5)\leftrightarrow(b^2,5)$, $(b^1,7)\leftrightarrow(c^1,7)$ şi $(a^2,15)\leftrightarrow(c^1,15)$. Dacă folosim exprimarea matematică fără precizarea momentelor de apariţie a alarmelor, indicând numai ordinea de apariţie, se va pierde caracteristica de simultaneitate a acestor alarme şi, în acest caz, reprezentarea matematică ar fi incompletă. Pentru a evidenţia faptul că anumite alarme au apărut simultan în jurnalul de alarme din expresia ***(2.1)***, vom folosi notaţia echivalentă:

$$J = \left\{ a^1 c^1 \begin{matrix} a^1 b^1 \\ b^2 c^1 \end{matrix} a^1 c^2 b^1 \begin{matrix} c^1 \\ a^2 \end{matrix} c^1 \right\} \qquad (2.4)$$

Expresia *(2.4)* constituie reprezentarea jurnalului de alarme *(2.1)* fără *constrângeri temporale* [70], deci fără precizarea momentelor de apariţie a alarmelor, dar cu respectarea ordinei de apariţie a alarmelor.

Prin extinderea ferestrei de observaţie asupra elementului de reţea UNIT1 momentele de apariţie ale alarmelor referitoare la acesta se reprezintă, în mod uzual, printr-un ansamblu de "benzi" de momente de apariţie. De exemplu în *Figura 2.5* se reprezintă 3 benzi de momente de apariţie a unor alarme:

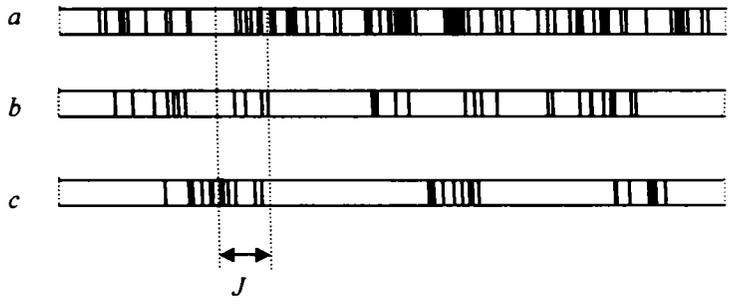


Figura 2.5 Reprezentarea alarmelor sub formă de benzi de momente de apariţie

Momentele de apariţie a alarmelor sunt definite ca primitive temporale din algebra momentelor [83]. Timpul este considerat un ansamblu discret total ordonat de momente. Astfel, pentru două momente t_1 şi t_2, constrângerea temporală, definită ca fiind de la t_1 la t_2, este reprezentată de un interval de timp $[I^-, I^+]$ care verifică relaţia:

$$t_1 + I^- \le t_2 \le t_1 + I^+ \qquad (2.5)$$

I^- şi I^+ definesc *minorantul* şi respectiv *majorantul* distanţei temporale orientate de la t_1 la t_2. Un graf T de constrângeri temporale între instanţe este un graf orientat ale cărui vârfuri sunt instanţele de timp, iar arcul de la t_1 la t_2 se numeşte *constrângerea temporală* de la t_1 la t_2 şi se notează $K_T(t_1 \to t_2)$. Absenţa constrângerilor temporale între două instanţe se reprezintă $(-\infty, +\infty)$.

Între grafurile constrângerilor temporale [78] vom defini relaţia de incluziune ca fiind o relaţie de ordine parţială, prin următoarea expresie matematică:

$$T_1 \subseteq T_2 \equiv_{def} \forall (t_1,t_2) \in T_1 \Rightarrow K_{T_1}(t_1 \rightarrow t_2) \subseteq K_{T_2}(t_1 \rightarrow t_2) \quad (2.6)$$

Un graf de constrângeri temporale poate avea mai multe reprezentări echivalente, dar există o singură reprezentare minimală a cărei calculare şi verificare a consistenţei globale se pot determina folosind algoritmul Floyd-Warshall [78], având o anumită complexitate. Implementarea algoritmului Floyd-Warshall în pseudo-limbaj de programare este următoarea :

```
function floyd-warshall (int[0..n,0..n] graph)
{
    // Initializarea grafului
    var int[0..n,0..n] dist := graph
    var int[0..n,0..n] pred
    for i = 0 to n
        for j = 0 to n
            if dist[i,j] > 0
                pred[i,j] := i
    // Bucla principală a algoritmului
    for k = 0 to n
        for i = 0 to n
            for j = 0 to n
                if dist[i,j] > dist[i,k] + dist[k,j]
                    dist[i,j] = dist[i,k] + dist[k,j]
                    pred[i,j] = pred[k,j]
    return dist
}
```

În continuare, vom considera sistematic reprezentarea minimală a grafului de constrângeri temporale.

2.3.2 Duratele alarmelor

Deoarece vom utiliza algebra momentelor, introdusă în lucrarea lui McDermott [83], pentru a introduce noţiunea de durată a alarmelor trebuie să facem un artificiu de reprezentare. Astfel, se poate introduce noţiunea de durată asociată alarmelor utilizând două momente pentru a defini alarma :

- Momentul de iniţiere sau *începutul* alarmei, notat t_0,
- Momentul de terminare sau *sfârşitul* alarmei, notat t_f.

Alarmele se exprimă, în acest caz, ca fiind triplete de elemente (x^i,t_0,t_f), unde x^i este tipul alarmei, referitor la elementul de reţea i, iar t_0 şi t_f sunt momentele de

început şi respectiv de sfârşit al alarmei. Astfel, se poate determina durata alarmei, folosind relaţia:

$$\delta(x^1) = t_f - t_0 \qquad (2.7)$$

Introducem ipoteza că alarmele de tipul a^1 (IN SERVICE) sunt *disjuncte* cu alarmele de tipul c^1 (OUT OF SERVICE). Explicaţia logică a acestei ipoteze este evidentă, starea de funcţionare fiind exclusivă cu starea de nefuncţionare. De asemenea, presupunem că un echipament în regim de funcţionare în suprasarcină este încă în stare de funcţionare, deci alarmele de tipul b^1 (OVERLOAD) se suprapun alarmelor de tipul a^1 (IN SERVICE).

Folosind notaţiile de triplete pentru a exprima alarmele, jurnalul de alarme din relaţia *(2.2)*, referitor la elementul de reţea UNIT1, se poate exprima astfel:

$$\hat{J}^1 = \{(a^1,1,4)(c^1,4,5)(a^1,5,7)(b^1,7,7)(c^1,7,10)(a^1,10,14)\\ (b^1,13,14)(c^1,14,15)(c^1,15,16)\} \qquad (2.8)$$

Reprezentarea grafică a duratelor alarmelor pentru jurnalului de alarme din relaţia ***(2.6)***, conform ipotezelor de lucru prezentate mai sus, referitor la elementul de reţea UNIT1 este cea din *Figura 2.6*:

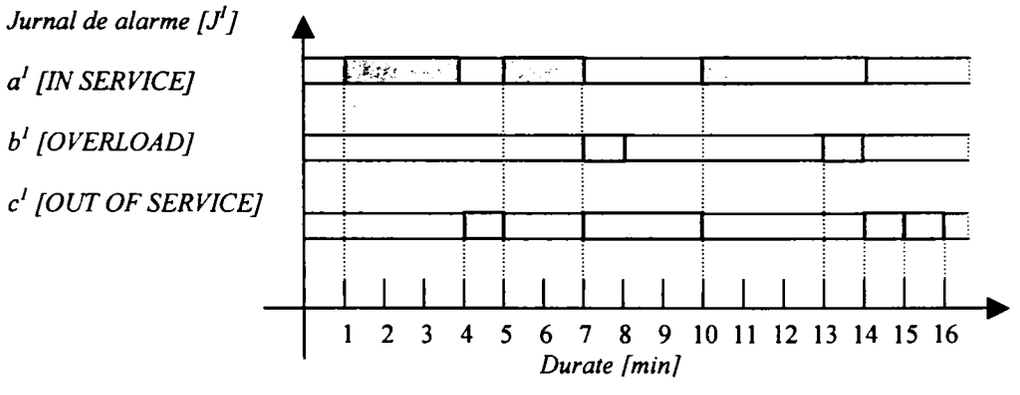


Figura 2.6 Reprezentarea grafică a duratelor alarmelor

Analizând reprezentarea de mai sus, obţinem un prim set de caracteristici ale alarmelor pentru fereastra de observaţie considerată:

- Calculând, cu relaţia ***(2.7)***, duratele alarmelor din relaţia ***(2.8)***, observăm că alarma a^1 durează minim 2 momente $(a^1,5,7)$ şi maxim 4 momente $(a^1,10,14)$. O primă concluzie este că echipamentul observat este într-un regim

de funcționare instabil, deoarece nu rămâne în serviciu mai mult de 4 minute, cel puțin pe durata perioadei de observație considerată.

- Duratele alarmelor b^1 se determină, în mod similar, ca fiind minim 0 minute și maxim 1 minut, conform expresiilor $(b^1,7,7)$ și $(b^1,13,14)$.
- Duratele alarmelor c^1 sunt minim 1 minut și maxim 3 minute, conform expresiilor $(c^1,4,5)$ respectiv $(c^1,7,10)$.

2.4 Cronici de alarme

Prin definiție, o *cronică de alarme* este o succesiune cronologică de alarme care poate fi evidențiată într-un jurnal de alarme. Într-o fereastră de observație a unui jurnal de alarme, cronicile pot să apară o singură dată sau de mai multe ori.

Cronicile de alarme alcătuite din două alarme se numesc cronici de ordinul 2. În mod similar, cronicile alcătuite din n alarme se numesc cronici de ordinul n. Evident, ordinul maxim posibil al unei cronici dintr-un jurnal de alarme este dat de lungimea jurnalului respectiv. De exemplu, pentru jurnalul de alarme din relația ***(2.2)***, ordinul maxim posibil al unei cronici este 9, deoarece acest jurnal este descris de o succesiune de 9 alarme : *acabcabcc*.

Pentru a determina, de exemplu, cronicile de ordinul 2 ale jurnalului de alarme din relația ***(2.2)***, vom rescrie jurnalul astfel:

$$J = \{(a,1)(c,4)(a,5)(b,7)(c,7)(a,10)(b,13)(c,14)(c,15)\} \qquad \textbf{(2.9)}$$

Combinațiile posibile de cronici de ordinul 2 sunt următoarele: $[aa],[bb],[cc],[ab],[ba],[ac],[ca],[bc]$ și $[cb]$.

Să analizăm succesiunea de alarme $[aa]$: pot fi evidențiate următoarele 3 apariții ale acestei cronici în jurnalul J: $(a,1)(a,5)$, $(a,1)(a,10)$ și $(a,5)(a,10)$. Intervalul minim de timp pentru cronica $[aa]$ este de 4 momente și corespunde succesiunii $(a,1)(a,5)$. Intervalul maxim de timp pentru cronica $[aa]$ este de 9 momente și corespunde succesiunii $(a,1)(a,10)$. Aceste observații se pot reprezenta grafic ca în *Figura 2.7* :

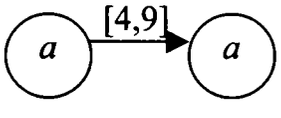


Figura 2.7 Exemplu de cronică de alarme

Folosind aceiaşi metodă, putem reprezenta grafic cronicile de alarme $[bb],[cc],[ab],[ba],[ac],[ca],[bc]$ şi $[cb]$ ca în *Figura 2.8*:

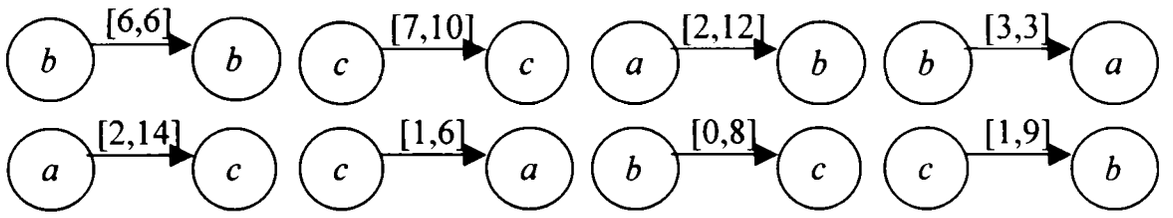


Figura 2.8 Cronici de alarme de ordinul 2

În acest exemplu, am reprezentat toate cronicile posibile de ordinul 2 pentru jurnal de alarme considerat, timp de 15 momente (minute). În situaţia în care jurnalul de alarme ar avea o fereastră de observaţie mai mare de 15 momente, dar cu acelaşi număr de alarme, cronicile de alarme de ordinul 2 ar putea rămâne neschimbate. Bazat pe o analiză a frecvenţei de apariţie a acestor cronici de ordinul 2 sau de ordine superioare, vom putea prezenta sistemului de supervizare doar acele cronici care se repetă mai des.

Vom demonstra în paragraful §2.5 că soluţia determinată pentru exemplul considerat este o cronică de alarme frecventă care exprimă clar instabilitatea în funcţionare a elementului de reţea UNIT1 şi trebuie luată o decizie de intervenţie asupra acestui element.

2.4.1 Modele de cronici

În acest paragraf, introducem definiţiile şi notaţiile matematice pentru descrierea cronicilor de alarme. Noţiunile şi notaţiile folosite se regăsesc în lucrările [5], [6], [44], [45], [58] şi [131].

Definiţia 2.3

Un model de cronici, notat K, este definit ca fiind un cuplu (j,T) unde j este un ansamblu de alarme dintr-un jurnal de alarme J, iar T este un graf de constrângeri temporale între aceste alarme.

De exemplu, pentru jurnalul prezentat în *Figura 2.1*, dacă alegem ansamblul de alarme referitoare la elementul de reţea UNIT1 ca fiind $[a,b,c]$, atunci vom putea determina un model de cronici corespunzător acestor alarme, în raport cu graful de constrângeri temporale $T = \{1,4,5,7,10,13,14\}$.

Definim un model de cronici fără constrângeri dacă alarmele sale nu au nici o ordine de apariţie. Notarea acestor modele de cronici se poate face fără a mai preciza

momentele de apariţie a alarmelor. Astfel în loc de $\{(a,t_1)(b,t_2)(b,t_3)\}$ dacă nu avem nici o constrângere temporală între instanţele de timp $t_1,t_2,t_3 \in T$ atunci vom nota fie $\{abb\}$, fie $\{bab\}$, fie $\{bba\}$.

2.4.2 Instanţe de cronici

Pentru a introduce noţiunea de instanţă de cronici facem următoarea definiţie:

Definiţia 2.4

Instanţa k a unui model de cronici K este definită ca fiind un ansamblu de alarme care au specificate momentele lor de apariţie.

Specificarea momentelor de apariţie ale alarmelor se numeşte *instanţiere*. De exemplu $(a^1,5)(c^1,7)(b^1,7)$ este o instanţă a modelului de cronici din *Figura 2.1*.

În scopul modelării matematice, instanţele de cronici pot fi extinse cu definiţiile 2.5 şi 2.6 care introduc noţiunile de sub-instanţe şi, respectiv, de supra-instanţe de cronici:

Definiţia 2.5

O instanţă k se numeşte sub-instanţă a unei instanţe k^i dacă şi numai dacă ea este o submulţime a lui k^i. Relaţia între sub-instanţă şi instanţă se notează $k \subseteq k^i$.

Definiţia 2.6

O instanţă k se numeşte supra-instanţă a unei instanţe k^i dacă şi numai dacă k^i este o submulţime a lui k. Relaţia între supra-instanţă şi instanţă se notează $k \supseteq k^i$.

De exemplu, $(a^1,5)(b^1,7)$ este o sub-instanţă a lui $(a^1,5)(c^1,7)(b^1,7)$ dar nu este o sub-instanţă pentru $(a^1,10)(c^1,13)(b^1,14)$.

Dezvoltarea definiţiilor modelelor de cronici de alarme poate fi făcută prin introducerea noţiunilor de sub-model şi de supra-model de cronici, folosind definiţiile 2.7 şi, respectiv, 2.8:

Definiţia 2.7

Un model de cronici K este sub-model al unei cronici K^i dacă şi numai dacă pornind de la orice instanţă a lui K^i putem extrage o instanţă a lui K. Relaţia între sub-model şi model se notează $K \subseteq K^i$.

Definiția 2.8

Un supra-model, se definește în mod similar cu definiția 2.7, dacă și numai dacă se respectă relația inversă, folosind notația $K \supseteq K^i$.

Definițiile sub-modelelor și supra-modelelor de cronici introduc astfel noțiuni de ordine parțială între modelele de cronici.

2.4.3 Cronici frecvente

Pornind de definițiile cronicilor de alarme și ale sub-modelelor și supra-modelelor de cronici, prezentate în paragraful anterior, se poate enunța următoarea teoremă:

Teorema 2.1

Un model de cronici $K(S,T)$ *este un sub-model al unei cronici* $K^i(S^i,T^i)$ *dacă și numai dacă* $S \subseteq S^i$ *și* $T \supseteq T^i$.

Demonstrația acestei teoreme este prezentată în lucrarea [131].

Ideea principală pentru determinarea cronicilor care se repetă frecvent, într-un jurnal de alarme, este de a genera toate cronicile posibile, numite *cronici candidate*, calculând apoi frecvența lor de apariție și reținând doar cronicile candidate care se repetă cu o frecvență mai mare decât o frecvență minimă de apariție considerată. Aceste cronici rezultate sunt numite *cronici frecvente*.

Se poate enunța următoarea teoremă, demonstrată de asemenea în lucrarea [131]:

Teorema 2.2

Dacă o cronică de alarme dintr-un jurnal J *este frecventă, atunci toate sub-instanțele ei sunt, de asemenea, frecvente în acel jurnal* J.

Condiția necesară și suficientă pentru ca o cronică C^i să nu fie frecventă este dacă cel puțin una dintre sub-instanțele ei nu este frecventă. Menționăm faptul că atunci când toate sub-instanțele unei cronici sunt frecvente cronica nu este neapărat frecventă. Mai exact, chiar dacă toate sub-instanțele unei cronici sunt frecvente nu vom putea exprima sigur faptul că respectiva cronică este frecventă.

De exemplu, considerând cronica $[abc]$ și sub-instanțele ei $[a]$, $[b]$, $[c]$, $[ab]$, $[bc]$ și $[ac]$, putem enunța că dacă una din sub-instanțe nu este frecventă atunci sigur cronica $[abc]$ nu este frecventă. Pe de altă parte, dacă toate sub-instanțele lui $[abc]$ sunt frecvente, nu putem concluziona despre $[abc]$ decât că ar putea fi frecventă.

Într-o altă ordine de idei, considerând cronica [*ab*] frecventă, deducem că [*a*] şi [*b*] sunt frecvente. În raport cu cronica [*abc*] mai rămâne de verificat frecvenţa lui [*bc*] şi [*ac*], care sunt sub-instanţe de ordinul *i-1*, pentru a putea determina dacă [*abc*] nu este frecventă.

2.5 Algoritm pentru recunoaşterea cronicilor frecvente

În continuare se prezintă algoritmul simplu de recunoaştere a cronicilor frecvente, introdus în lucrările [46], [48], [58] şi [131].

Pentru a construi un algoritm în scopul recunoaşterii cronicilor frecvente într-un jurnal de alarme trebuie să pornim de la un nivel iniţial al alarmelor unitare şi distincte care intervin în acel jurnal şi să stabilim pentru început o frecvenţă minimă de apariţie a acestor alarme în jurnalul considerat.

La prima iteraţie a algoritmului vom considera cronicile de ordinul 1, care sunt deci chiar alarmele componente ale jurnalului. Vom reţine dintre alarmele componente doar acele alarme care au o frecvenţă de apariţie mai mare decât o valoare de limită minimă considerată, notată $f_{\min}$.

La iteraţia următoare vom construi cronicile de ordinul 2 pornind de la alarmele frecvente reţinute la iteraţia anterioară. Cronicile de ordinul 2 vor fi denumite cronici candidate şi asupra lor trebuie aplicată o formulă de calcul pentru a determina frecvenţa lor de apariţie în jurnalul considerat. Vom reţine pentru iteraţia următoare doar acele cronici candidate care depăşesc limita considerată $f_{\min}$.

La fiecare nouă iteraţie i a algoritmului, se generează cronici candidate de ordinul i pornind de la cronicile frecvente de ordinul i-1, care au fost în prealabil determinate la iteraţia anterioară. Apoi se calculează frecvenţele de apariţie ale cronicilor candidate şi se elimină acele cronici candidate care au o frecvenţă de apariţie mai mică decât $f_{\min}$.

Generarea cronicilor candidate de ordinul *i*, pornind de la cronicile frecvente determinate la iteraţia anterioară a algoritmului, poate fi realizată în mai multe moduri. În paragraful următor voi prezenta modurile de generare de cronici candidate prin asamblare paralelă şi prin asamblare serială.

În principiu, asamblarea paralelă presupune respectarea unei anumite ordini între alarme, adăugând câte o alarmă într-o poziţie corespunzătoare ordinii prestabilite în cadrul cronicii de la care se porneşte generarea. Ordinea prestabilită este definită de către expert pe baza unui raţionament specific. De exemplu, se poate lua în considerare, în cazul supervizării locale a alarmelor, gradul de prioritate. Dacă se cunoaşte că o alarmă *a* este mai prioritară decât o alarmă *b*, atunci se poate considera ordinea $a<b$ pentru analiza alarmelor *a* şi *b*.

Asamblarea serială nu ţine cont de ordinea alarmelor, adăugând câte o alarmă în poziţia finală a cronicii de la care se porneşte generarea.

Algoritmul se va opri la iteraţia $i+j$ pentru care nu mai există nici o cronică frecventă de ordinul $i+j$. Menţionez că ordinul maxim al cronicilor frecvente este în mod evident egal cu dimensiunea totală a jurnalului J, deoarece se poate ajunge, în cazul cel mai favorabil, să se construiască o cronică alcătuită din totalitatea alarmelor care intervin. Ordinul n al unei cronici de alarme dintr-un jurnal J este deci $n \le \dim|J|$, $\dim|J|$ fiind dimensiunea totală a jurnalului J.

Fiind dat un jurnal de alarme J şi un proces de recunoaştere de cronici de alarme, vom nota $A(J)$ ansamblul de cronici candidate şi $\overline{A}(J)$ ansamblul de cronici frecvente recunoscute în jurnalul J. Este evident că rezultatul determinării cronicilor frecvente depinde de strategia de generare a cronicilor candidate care este folosită (asamblare paralelă sau serială de alarme).

Algoritm pentru recunoaşterea cronicilor frecvente într-un jurnal de alarme se poate reprezenta folosind schema logică din *Figura 2.9*:

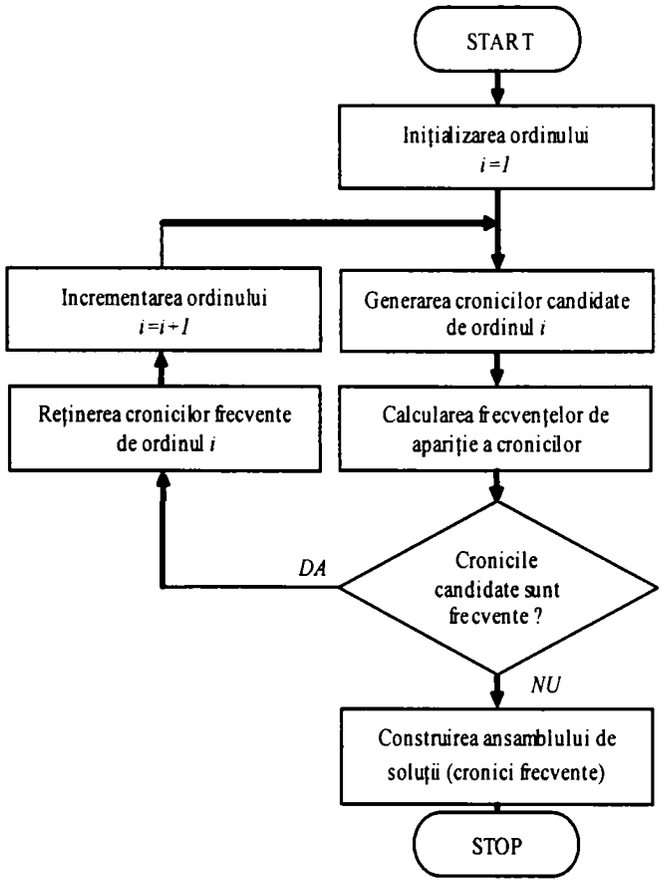


Figura 2.9 Algoritm pentru recunoaşterea cronicilor frecvente

Algoritmul prezentat în *Figura 2.9* se poate exprima în pseudo-limbaj de programare, astfel:

Algoritm *Recunoaştere_cronici_frecvente*(J , $f_{\min}$)

Intrări J =jurnalul de alarme, $f_{\min}$ =frecvenţa minimă de apariţie

Ieşire $\overline{A}(J)$ = mulţimea cronicilor frecvente din jurnalul J

/* Iniţializarea ordinului i la 1 */

$i \leftarrow 1$;

/* Buclă secvenţială **do until** mulţimea cronicilor frecvente */

/* de ordinul i este vidă */

Do

/* Generarea cronicilor candidate de ordinul i */

$$A_i(J) \leftarrow \left\{ \bigcup_{i}^{\dim|a_i|=i} a_i \middle| a_i \in Candidate\_[p|s](\overline{A}_i(J)) \right\};$$

/* Calcularea frecvenţelor minime de apariţie şi reţinerea */

/* doar a cronicilor frecvente de ordinul i */

$$\overline{A}_i(J) \leftarrow \left\{ \bigcup_{i}^{\dim|a_i|=i} a_i \middle| a_i \in A_i(J), f(a_i) \geq f_{\min} \right\};$$

/* Incrementarea ordinului i */

$i \leftarrow i+1$;

/* Sfârşit buclă secvenţială */

Until $\overline{A}_i(J) = NULL$;

/* Construire rezultat : mulţimea tuturor cronicilor frecvente */

/* de la ordinul 1 la $i-1$ din jurnalul de alarme J */

Return $\overline{A}(J) \leftarrow \bigcup_{n=1}^{i-1} \overline{A}_n(J)$;

Pentru exemplificarea procedurii descrise mai sus, să determinăm cronicile frecvente din jurnalul de alarme exprimat prin relaţia ***(2.2)***. Pentru început, scriem relaţia ***(2.2)*** într-o formă echivalentă:

$$J = \{(a,1)(c,4)(a,5)(b,7)(c,7)(a,10)(b,13)(c,14)(c,15)\} \qquad \textbf{(2.10)}$$

Deoarece jurnalele de alarme pot fi exprimate fără a indica momentele de apariţie a alarmelor, însă păstrând caracteristica de simultaneitate a anumitor alarme, jurnalul de alarme de mai sus se poate exprima astfel:

$$J = \left\{ aca \begin{matrix} b \\ c \end{matrix} abcc \right\} \qquad (2.11)$$

Frecvenţa minimă a cronicilor fără constrângeri temporale se calculează folosind următoarea relaţie:

$$f_{\min}(k) = \left[\min_{k \in J} \left(\frac{|x^i \in J|}{|x^i \in k|} \right) \right] \qquad (2.12)$$

În relaţia de mai sus, $|x^i \in J|$ reprezintă numărul de alarme de tipul x^i din jurnalul J, iar $|x^i \in k|$ reprezintă numărul de alarme de tipul x^i din cronica respectivă k.

Considerăm frecvenţa minimă de apariţie $f_{\min} = 2$, pentru cronicile care vor fi determinate în acest jurnal J din relaţia **(2.11)**.

De exemplu, calculul frecvenţei minime a cronicii abc din jurnalul exprimat prin relaţia **(2.11)** este:

$$f_{\min}(abc) = \left[\min \left(\frac{|a \in J|}{|a \in [abc]|} = \frac{3}{1}, \frac{|b \in J|}{|b \in [abc]|} = \frac{2}{1}, \frac{|c \in J|}{|c \in [abc]|} = \frac{4}{1} \right) \right] = 2 \qquad (2.13)$$

La prima iteraţie a algoritmului se obţine mulţimea totală a alarmelor (cronici de ordinul 1) care apar în jurnalul J. Generarea cronicilor de ordinul 1 este de fapt echivalentă cu generarea tuturor alarmelor care apar în jurnal. Se vor reţine doar acele alarme care sunt frecvente. Deoarece a se produce de 3 ori, b de 2 ori şi c de 4 ori, toate aceste cronici de ordinul 1 sunt frecvente faţă de limita considerată, deci:

$$\overline{A}_1(J) = \{a \quad b \quad c\} \qquad (2.14)$$

Condiţia de intrare în bucla secvenţială a procedurii este îndeplinită, mulţimea rezultată nefiind vidă. Menţionez că dacă s-ar alege o frecvenţă minimă prea mare, de exemplu f_{min}=5, atunci nici una dintre alarmele prezentate nu ar fi frecvente şi algoritmul s-ar opri la această primă iteraţie.

Mulţimea de cronici candidate se obţine prin combinarea alarmelor care intervin, pentru a obţine cronici candidate de ordinul 2. Prin asamblarea paralelă, respectând ordinea prestabilită a b c, se generează în iteraţia a 2-a următoarele 6 cronici candidate:

$$A_2(J) = \{aa \quad ab \quad ac \quad bb \quad bc \quad cc\} \tag{2.15}$$

Calculul frecvenţelor minime ale cronici candidate, pornind de la relaţia (1.12), determină următorul rezultat:

$$f_{\min}[A_2(J)] = \left\{\frac{3}{2} \quad 2 \quad 3 \quad 1 \quad 2 \quad 2\right\} \tag{2.16}$$

Păstrăm numai cronicile candidate frecvente, având frecvenţa minimă de apariţie 2, obţinem mulţimea de cronici frecvente de ordinul 2:

$$\overline{A}_2(J) = \{ab \quad ac \quad bc \quad cc\} \tag{2.17}$$

La următoarea trecere prin bucla secvenţială, se determină cronicile candidate de ordinul 3 care se construiesc prin asamblare paralelă astfel:

$$\begin{array}{c|c|c|c|c|c|c|c|c} & a & aab & a & aac & a & abc & a & acc \\ ab & b \Rightarrow & abb, ac & b \Rightarrow & abc, bc & b \Rightarrow & bbc, cc & b \Rightarrow & bcc \\ & c & abc & c & acc & c & bcc & c & ccc \end{array} \tag{2.18}$$

Mulţimea cronicilor candidate de ordinul 3 este alcătuită din următorul ansamblu de cronici:

$$A_3(J) = \{aab \quad abb \quad abc \quad aac \quad acc \quad bbc \quad bcc \quad ccc\} \tag{2.19}$$

Calculând frecvenţele minime de apariţie asociate acestor cronici candidate, obţinem următoarea mulţime de valori:

$$f_{\min}[A_3(J)] = \left\{\frac{3}{2} \quad 1 \quad 2 \quad \frac{3}{2} \quad 2 \quad 1 \quad 2 \quad \frac{4}{3}\right\} \tag{2.20}$$

Păstrăm doar cronicile candidate frecvente de ordinul 3:

$$\overline{A}_3(J) = \{abc \quad acc \quad bcc\} \tag{2.21}$$

Cronicile candidate de ordinul 4 se determină, în mod similar:

$$abc\begin{vmatrix} a \\ b \\ c \end{vmatrix} \Rightarrow \begin{vmatrix} aabc \\ abbc \\ abcc \end{vmatrix}, acc\begin{vmatrix} a \\ b \\ c \end{vmatrix} \Rightarrow \begin{vmatrix} aacc \\ abcc \\ accc \end{vmatrix}, bcc\begin{vmatrix} a \\ b \\ c \end{vmatrix} \Rightarrow \begin{vmatrix} abcc \\ bbcc \\ bccc \end{vmatrix} \quad (2.22)$$

Ansamblul de cronici candidate de ordinul 4 este următorul:

$$A_4(J) = \{aabc \quad abbc \quad abcc \quad aacc \quad accc \quad bbcc \quad bccc\} \quad (2.23)$$

Frecvențele minime, calculate pentru aceste cronici candidate de ordinul 4, sunt următoarele:

$$f_{\min}[A_4(J)] = \left\{\frac{3}{2} \quad 1 \quad 2 \quad \frac{3}{2} \quad \frac{4}{3} \quad 1 \quad \frac{4}{3}\right\} \quad (2.24)$$

Reținem numai cronicile frecvente cu frecvența minimă 2, în acest caz, rezultând o singură cronică frecventă:

$$\overline{A}_4(J) = \{abcc\} \quad (2.25)$$

La următoarea trecere prin bucla secvențială a procedurii, vom observa că nu mai există cronici frecvente de ordinul 5. Mulțimea cronicilor candidate de ordinul 5 este:

$$A_5(J) = \{aabcc \quad abbcc \quad abccc\} \quad (2.26)$$

Frecvențele minime calculate, pentru aceste cronici candidate de ordinul 5, sunt toate sub frecvența minimă considerată:

$$f_{\min}[A_5(J)] = \left\{\frac{3}{2} \quad 1 \quad \frac{4}{3}\right\} \quad (2.27)$$

Procedura de căutare a cronicilor frecvente se va opri aici, deoarece rezultatul cronicilor de ordinul 5 este mulțimea vidă.

În acest caz, deoarece nu mai există nici o cronică frecventă de ordinul 5, pentru frecvența minimă $f_{\min} = 2$, în jurnalul $J = \left\{aca\begin{matrix} b \\ c \end{matrix} abcc\right\}$, rezultatul algoritmului se

construieşte ca fiind mulţimea tuturor cronicilor frecvente, de la ordinul 1 până la ordinul 4, astfel:

$$\overline{A}(J)=\{a \quad b \quad c \quad ab \quad ac \quad bc \quad cc \quad abc \quad acc \quad bcc \quad abcc\} \qquad \textbf{(2.28)}$$

Rezultatul procedurii de mai sus, referitoare la jurnalul de alarme din expresia ***(2.11)***, poate fi reprezentat grafic ca în *Figura 2.10*:

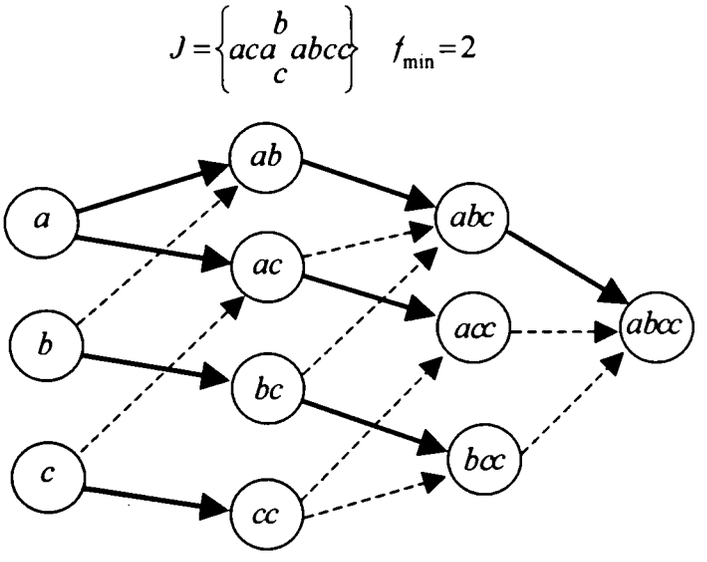


Figura 2.10 Cronici frecvente prin asamblare paralelă

În reprezentarea grafică din *Figura 2.10* am indicat construirea directă a cronicilor candidate prin linii pline şi construirea indirectă a cronicilor prin linii punctate. Această reprezentare grafică intuitivă va fi dezvoltată în paragraful §6.7.2.

2.5.1 Proceduri pentru generarea cronicilor candidate

După cum se poate observa din modul de asamblare a cronicilor candidate din paragraful anterior, am folosit tehnicile de construire a candidatelor în spaţiul cronicilor paralele.

Spaţiul cronicilor paralele se utilizează pentru a evita problema de multiplicare a cronicilor. În acest caz, se adaugă alarme respectând o ordine predefinită între tipurile de alarme. Cronicile candidate se obţin adăugând alarme de tip superior sau egal celui mai mare tip de alarme din ansamblul de cronici. De exemplu, *ab* generează *abb* şi *abc* şi nu mai trebuie generat *aab* deoarece această cronică s-ar obţine din cronica *aa* care nu este frecventă.

O caracteristică principală a spațiului cronicilor paralele este că, de exemplu, cronicile ab şi ba sunt, de fapt, reduse la aceeaşi cronică paralelă, respectând ordinea predefinită între tipurile de alarme a şi b.

Ipoteza de lucru din spațiul cronicilor paralele, pentru a simplifica gradul de complexitate a problemei de generare a cronicilor candidate, este că toate instanțele lui $\overline{A}_i(J)$ trebuie să fie disjuncte două câte două.

Procedura de determinare a cronicilor frecvente prin asamblare paralelă se scrie astfel, în pseudo-limbaj de programare:

```
Procedură   Candidate_p( A_n(J) )
/* Generare cronici candidate prin asamblare paralelă */
Intrare     A_n(J) mulțimea cronicilor frecvente de ordinul n
Ieşire      A_{n+1}(J) mulțimea cronicilor candidate de ordinul n+1
/* Inițializarea mulțimii de cronici candidate de ordinul n+1 */
      A_{n+1}(J) = NULL;
      For ∀a_1a_2...a_i ∈ A_n(J)
            For ∀a_j ∈ J AND a_j ≥ a_k, ∀k ∈ [1..n]
/* Adăugare alarme de tip superior sau egal */
                  Cronica_paralelă_candidată = a_1a_2...a_ia_j ;
/* Verificarea sub-cronicilor pentru cronica paralelă candidată */
                  If toate sub-instanțele lui a_1a_2...a_ia_j ∈ A_n(J)
                        A_{n+1}(J) = A_{n+1}(J) ∪ a_1a_2...a_ia_j ;
                  End If;
            End For;
      End For;
/* Rezultatul este mulțimea cronicilor candidate de ordinul n+1 */
      Return A_{n+1}(J) ;
```

Pentru determinarea cronicilor candidate se poate considera, de asemenea, spațiul cronicilor seriale. În acest caz nu se respectă o anumită ordine între tipurile de alarme, doar se adaugă la sfârşitul candidatei câte una dintre alarmele din jurnal.

Procedura în pseudo-limbaj de programare, corespunzătoare determinării cronicilor candidate prin asamblare serială, se scrie astfel:

```
Procedură   Candidate_s( A_n(J) )
/* Generare cronici candidate prin asamblare seriala */
```

Intrare $\overline{A}_n(J)$ mulţimea cronicilor frecvente de ordinul n

Ieşire $A_{n+1}(J)$ mulţimea cronicilor candidate de ordinul $n+1$

/* Iniţializarea mulţimii de cronici candidate de ordinul $n+1$ */

$A_{n+1}(J) = NULL;$

For $\forall a_1 a_2 ... a_i \in \overline{A}_n(J)$

For $\forall a_j \in J$

/* Adăugare alarme la sfârşitul cronicii candidate */

Cronica_serială_candidată = $a_1 a_2 ... a_i a_j$;

/* Verificarea sub-cronicilor pentru cronica serială candidată */

If toate sub-instanţele lui $a_1 a_2 ... a_i a_j \in \overline{A}_n(J)$

$A_{n+1}(J) = A_{n+1}(J) \bigcup a_1 a_2 ... a_i a_j$;

End If;

End For;

End For;

/* Rezultatul este mulţimea cronicilor candidate de ordinul $n+1$ */

Return $A_{n+1}(J)$;

Rezultatul procedurilor de mai sus, conţinând determinarea cronicilor frecvente în spaţiul cronicilor seriale, referitor la jurnalul de alarme din expresia ***(2.11)***, poate fi reprezentat grafic ca în *Figura 2.11*:

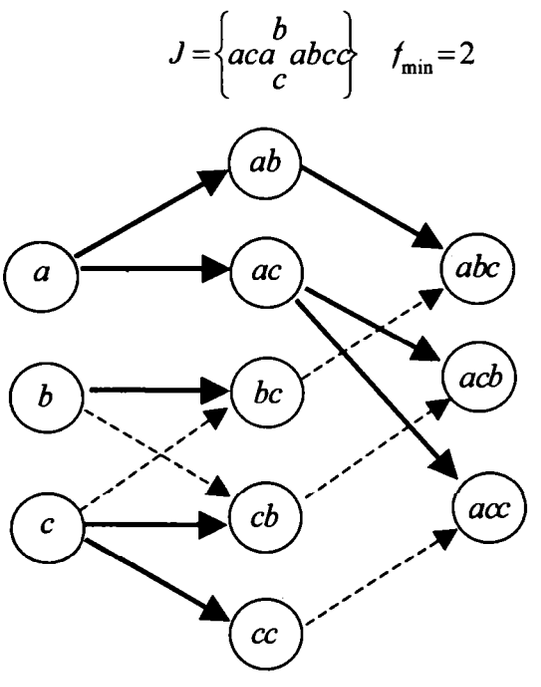


Figura 2.11 Cronici frecvente prin asamblare serială

Fiind date două alarme a şi b, constrângerile temporale se determină din analiza momentelor de apariţie a acestor alarme în jurnalul de alarme $J = \{(a,1)(c,4)(a,5)(b,7)(c,7)(a,10)(b,13)(c,14)(c,15)\}$.

În primul rând, se determină mulţimea aparţiilor celor două alarme, notată $A(a,b)$, ca fiind un ansamblu al tuturor aparţiilor de alarme a şi b în jurnalul J:

$$A(a,b) = \{[(a,1)(b,7)], [(a,1)(b,13)], [(a,5)(b,7)], [(a,5)(b,13)], [(b,7)(a,10)], [(a,10)(b,13)]\} \quad (2.29)$$

Pornind de la mulţimea aparţiilor celor două alarme $A(a,b)$, se pot calcula constrângerile temporale între a şi b ca fiind limitele superioare ale distanţei temporale de la a la b. Astfel, apariţia $[(a,t_a)(b,t_b)]$ permite calcularea constrângerii temporale $I = t_b - t_a$.

Definim pentru fiecare constrângere temporală I o variabilă $c(I)$ care este raportul dintre numărul de aparţii ale lui a şi b, care verifică expresia I, faţă de numărul total de aparţii din $A(a,b)$.

$$c(I) = \frac{|\{[(a,t_a)(b,t_b)] \in A(a,b), t_b - t_a = I\}|}{|A(a,b)|} \quad (2.30)$$

De exemplu, există 3 aparţii ale lui a şi b care verifică $I = [0,6]$, acestea fiind $[(a,1)(b,7)]$, $[(a,5)(b,7)]$ şi $[(a,10)(b,13)]$, astfel se determină $c([0,6]) = \frac{3}{6} = 0,5$.

Există 5 aparţii ale lui a şi b care verifică $I = [-3,3] \cup [6,8]$, acestea fiind $[(a,1)(b,7)]$, $[(a,5)(b,7)]$, $[(a,5)(b,13)]$, $[(b,7)(a,10)]$ şi $[(a,10)(b,13)]$, astfel $c([-3,3] \cup [6,8]) = \frac{5}{6} = 0,83$.

Pentru $I = [-3,12]$ toate cele 6 aparţii verifică I şi deci $c([-3,12]) = \frac{6}{6} = 1$.

O constrângere temporală I se numeşte *constrângere globală* dacă $c(I) = 1$. Dacă nu se verifică $c(I) = 1$, atunci constrângerea temporală I se numeşte *constrângere locală.*

Pentru $c(I)$ se pot determina următoarele proprietăţi, demonstrate în lucrarea[10]:

$$\forall I, 0 \le c(I) \le 1 \qquad (2.31)$$

$$\forall I, \forall I', I \subseteq I' \Rightarrow c(I) \le c(I') \qquad (2.32)$$

$$\forall I, \forall I', I \cap I' = \emptyset \Rightarrow c(I) + c(I') = c(I \cup I') \qquad (2.33)$$

Pentru a explica procedura de determinare a constrângerilor între două alarme, prezentăm căutarea constrângerii minime $c(I) = 0{,}6$ pentru aparițiile lui a şi b din relația ***(2.33)***. Pentru început, determinăm $n = 6$ ca fiind numărul de apariții din $A(a,b)$.

Spațiul constrângerilor temporale dintre a şi b este:

$$S = \{-3,2,3,6,8,12\} \qquad (2.34)$$

Mulțimea constrângerilor temporale dintre a şi b este:

$$C(I) = \{[-3,6],[-3,8],[-3,12],[2,8],[2,12],[3,12]\} \qquad (2.35)$$

2.6.1 Procedura de determinare a constrângerilor temporale

În continuare se prezintă o procedură de calcul pentru analiza constrângerilor temporale dintre alarme care poate fi introdusă în cadrul algoritmului în scopul analizei relațiilor dintre alarmele care constituie cronicile candidate.

Procedura de determinare a constrângerilor temporale minime între două alarme se exprimă în pseudo-limbaj de programare astfel:

Procedură *Constraint*(a , b , $c_{\min}(I)$)
/* determinare constrângeri temporale */
Intrare $A(a,b) = \{[(a,t_a^i),(b,t_b^i)], i = 1..n\}$ mulțimea aparițiilor alarmelor a şi b, $c_{\min}(I)$ constrângerea minimă.
Ieşire $C(I)$ mulțimea constrângerilor care respectă $c_{\min}(I)$
/* Inițializarea mulțimii constrângerilor temporale */
$C(I) = NULL;$
/* Inițializarea spațiului constrângerilor temporale */
$S \leftarrow \{t_b^i - t_a^i | i = 1..n\};$
/* Constanta temporală k */
$k \leftarrow [c_{\min}(I) * n];$
/* Sortarea spațiului constrângerilor temporale */
Sorting $S = \{x_1 \le ... \le x_n\};$

```
      For i = 1 ≤ n − k + 1
            For j = i + k − 1 ≤ n
/* Determinarea constrângerilor temporale care respectă c_min(I) */
                  C(I) = C(I) ∪ {[x_i, x_j]};
            End For;
      End For;
/* Rezultatul este mulţimea constrângerilor temporale C(I) */
      Return C(I);
```

Această procedură de determinare a constrângerilor temporale poate fi folosită pentru a exprima condiţii suplimentare de lucru pentru procedurile de generare de cronici candidate în scopul determinării cronicilor frecvente.

2.7 Concluzii

În acest capitol am prezentat procedurile pentru determinarea cronicilor frecvente din jurnale de alarme, folosind un principiu de calcul în două etape : generarea unor cronici candidate şi apoi reţinerea acelor cronici candidate care sunt frecvente. Metoda prezentată poate fi aplicată în spaţiul cronicilor seriale sau paralele, cu sau fără constrângeri temporale.

Observaţie asupra complexităţii procedurii de construire a cronicilor candidate: în experimentele efectuate, ordinul n al cronicilor frecvente descoperite în jurnalele de alarme studiate nu depăşeşte ordinul unităţilor, fiind $n < 10$ în majoritatea cazurilor. Problema complexităţii exponenţiale a procedurilor prezentate apare atunci când creşte frecvenţa minimă a cronicilor candidate.

În raport cu metoda prezentată în lucrarea [80], am obţinut ordine de complexitate comparabile pentru procedurile prezentate. În plus, metoda prezentată permite introducerea noţiunilor de constrângeri temporale pentru a reduce complexitatea procedurilor.

Pentru a dezvolta procedurile studiate, vom introduce reprezentări bazate pe formalismul reţelelor Petri. Reţelele Petri vor permite descrierea detaliată a evoluţiei cronicilor prin modelarea procesului de generare a alarmelor componente.

În capitolul următor voi introduce câteva noţiuni despre teoriile temporale bazate pe intervale şi teoriile temporale bazate pe momente, pentru a explica de ce am ales algebra momentelor descrisă de McDermott [83] în cadrul modelării matematice pentru analiza alarmelor.

Capitolul 3. ALGEBRA TEMPORALĂ

3.1 Introducere

Timpul reprezintă un factor esențial în modelarea proceselor naturale, iar în literatura de specialitate s-a ajuns la cercetarea şi dezvoltarea mai multor teorii temporale. Principalele teorii temporale sunt prezentate în lucrările de referință [12], [13], [28], [40], [71], [74], [75] şi [129]. Obiectivul major al cercetărilor teoretice în domeniul timpului este de a determina elementele primare ale timpului în raport cu aplicația de modelare. Majoritatea teoriilor temporale dezvoltate sunt bazate pe axiome definite în logica predicatelor.

Sistemele temporale, introduse de teoriile temporale, definesc structuri asemănătoare dar sunt diferite din punct de vedere al formalismului utilizat. Cea mai importantă diferență între diversele teorii temporale este interpretarea intervalelor de timp. Astfel, teoriile temporale pot fi clasificate în algebra intervalelor, algebra momentelor şi algebra punctelor.

Mai multe teorii temporale, cum sunt cele propuse de Ladkin [74] şi [75], Dechter [40], Maiocchi [79] şi Bruce [28], au la bază puncte sau momente ca elemente primare. În aceste teorii, intervalele de timp sunt definite pornind de la puncte, în general prin declararea punctelor de început şi de sfârşit ale intervalelor. În acest caz, se pune problema dacă punctele astfel definite, numite şi puncte de frontieră ale intervalului, sunt incluse în interval sau trebuie adresate în alt mod.

Allen, în lucrările de referință [12] şi [13], propune definirea intervalelor închise şi deschise. Intervalele închise au proprietatea că două intervalele închise adiacente au un punct comun, iar intervalele deschise au proprietatea că două intervale deschise adiacente nu au nici un punct comun. Teoria temporală propusă de Allen tratează intervalele sub formă de primitive, iar punctele sub formă de locuri de întâlnire ale intervalelor.

În anumite teorii temporale, de exemplu în teoria propusă de Maiocchi [79], toate intervalele sunt declarate semi-deschise. Proprietatea principală a intervalelor semi-deschise este că acestea pot forma în mod convenabil alăturări de intervale adiacente.

Teoriile temporale propuse de Vilain [129] şi Knight-Ma [71] consideră atât punctele cât şi intervalele sub formă de primitive.

În lucrările de referință ale lui Beek [17] şi [18], se regăsesc definițiile pentru algebra intervalelor şi algebra punctelor, fiind prezentate, de asemenea, relațiile între intervale şi relațiile între puncte. O observație importantă relativă la definițiile din lucrările lui Beek este că aceste definiții nu conțin relațiile dintre intervale şi puncte.

Relațiile dintre intervale şi puncte sunt abordate în lucrarea lui Vilain [129], unde intervalele sunt definite în raport cu punctele şi cu ordinea corespondentă dintre puncte.

Teoriile temporale diferă şi în raport cu definirea linearităţii şi a densităţii elementelor de timp. Majoritatea sistemelor temporale presupun că timpul este liniar, astfel toate elementele de timp sunt ordonate pe parcursul unei linii temporale. În anumite sisteme temporale se propun variante neliniare.

De exemplu, în lucrarea lui McDermott [83] se propune o variantă neliniară a timpului care introduce algebra momentelor. În principiu, în logica temporală a algebrei momentelor, timpul se poate ramifica în viitor. Ramificarea timpului în viitor este descrisă de un set de axiome, care vor fi prezentate mai jos. Menţionăm aici că ramificarea timpului în viitor este importantă pentru modelarea unor sisteme de analiză referitoare la mai multe posibilităţi de evoluţie a unor evenimente.

Densitatea elementelor de timp depinde de tipul primitivelor propuse pentru definirea sistemului temporal. Astfel, pentru un sistem temporal care este bazat pe intervale, densitatea este definită în raport cu descompunerea intervalului, finită sau infinită. În sistemele temporale bazate pe puncte, densitatea este definită în raport cu proprietatea că, între oricare două puncte pe aceeaşi linie temporală, se poate găsi un al treilea punct intermediar.

Algebra momentelor presupune un sistem dens de puncte, pe când algebra punctelor definită de Knight-Ma [71] nu este densă, reprezentând un sistem discret unde fiecare element de timp are un unic predecesor şi un unic succesor. Algebra intervalelor propune, în general, o abordare mixtă de elemente de timp dense şi discrete.

O altă diferenţă majoră între teoriile temporale o constituie abilitatea acestora de a modela natura deschisă sau închisă a intervalelor. Sistemul temporal propus de Allen [12] şi [13] permite numai existenţa unor intervale de tip nedeterminat, deoarece nu sunt definite intervalele deschise sau închise, nefiind definit elementul de bază, punctul. Sistemul temporal propus de Vilain [129] descrie atât intervalele cât şi punctele ca fiind primitive, dar nu permite definirea intervalelor deschise sau închise.

Sistemul temporal propus de Knight-Ma [71] permite caracterizarea conceptelor convenţionale de intervale deschise, închise şi chiar a intervalelor semi-deschise şi semi-închise care sunt construite pe baza punctelor.

Importanţa definirii punctelor şi a intervalelor ca primitive este determinată de necesităţile de modelare unde trebuie aplicate teoriile temporale. Algebra intervalelor şi algebra momentelor oferă exemple de proprietăţi definite în timp şi majoritatea modelărilor matematice folosesc aceste sisteme temporale. În lucrarea lui Galton [55] se propune o extensie a algebrei intervalelor pentru a lua în considerare reprezentările de puncte pentru a descrie complet proprietăţile intervalelor de timp.

Vom prezenta definiţiile şi axiomele corespunzătoare diferitelor sisteme temporale, pentru a alege sistemul temporal cel mai adaptat modelării problemei de analiză a unor evenimente discrete ale unui sistem supervizat, cum sunt alarmele în cazul unui sistem de telecomunicaţii.

Distincția între puncte şi momente este dată tocmai de primitivele folosite pentru definirea acestora. Momentele sunt necesare pentru a modela complet un spațiu temporal dens sau discret. Axiomele prezentate pot fi descrise ca fiind extensii ale algebrei momentelor, pentru a include definiția punctelor prin primitive.

Una din principalele axiome ale algebrei momentelor, *Axioma 3.6*, care declară că momentele nu întâlnesc niciodată alte momente, conduce la concluzia că în algebra momentelor nu se ține cont de densitate. Această axiomă determină folosirea algebrei momentelor pentru modelarea momentelor de apariția a unor informații într-un sistem.

Axiomele generale pentru definirea timpului sunt independente de densitate sau de liniaritate. Acestor axiome li se alătură un set de axiome specifice pentru definirea densității şi a liniarității timpului.

Relațiile posibile, definite pornind de la intervale şi puncte, permit, de asemenea, introducerea noțiunii de instanțiere sau moment de apariție, care este folosită pentru managementul bazelor de date temporale [38], cum sunt jurnalele de alarme de telecomunicații.

3.2 Axiomele temporale bazate pe intervale

Teoria timpului bazată pe intervale este descrisă în lucrarea lui Allen [12] şi se bazează pe definirea unei clase *I* de intervale, care este axiomatizată pornind de la o unică relație temporală între intervale. Această unică relație temporală exprimă interdependența între intervale, mai exact dacă un interval *se întâlneşte* sau *nu se întâlneşte* cu un alt interval. Relația de întâlnire este definită prin primitiva *punct()*.

Operațiile matematice posibile definite în raport cu această primitivă sunt adiacența, notată ($\curlywedge$), şi disjuncția exclusivă, notată ($\neg$).

Setul de axiome pentru definirea algebrei intervalelor a fost prezentat pentru prima dată în lucrarea [14] şi a fost redefinit în lucrarea [15]. Acest set de 5 axiome este prezentat mai jos.

Axioma 3.1 declară că punctul temporal unde se întâlnesc două intervale este unic determinat şi este asociat acestor două intervale:

Axioma 3.1

$$\forall i,j,k,l \in I : punct(i,j) \curlywedge punct(i,k) \curlywedge punct(l,j) \Rightarrow punct(l,k) \qquad (3.1)$$

Axioma 3.2 are scopul de a asigura o ordine totală a punctelor de întâlnire ale intervalelor.

Axioma 3.2

$$\forall i,j,k,l \in I : punct(i,j) \wedge punct(k,l) \Rightarrow punct(i,l)$$
$$\neg \exists m \in I : punct(i,m) \wedge punct(m,l)$$
$$\neg \exists n \in I : punct(k,n) \wedge punct(n,j) \quad (3.2)$$

Axioma 3.3 defineşte cel puţin un interval precedent şi un interval succesiv:

Axioma 3.3

$$\forall i \in I, \exists j,k \in I : punct(j,i) \wedge punct(i,k) \quad (3.3)$$

Axioma 3.4 declară că între două puncte de întâlnire poate exista un singur interval.

Axioma 3.4

$$\forall j,k \in I : \exists i,l \in I \text{ a.î. } punct(i,j) \wedge punct(j,l) \wedge punct(i,k) \wedge punct(k,l)) \Rightarrow j = k \quad (3.4)$$

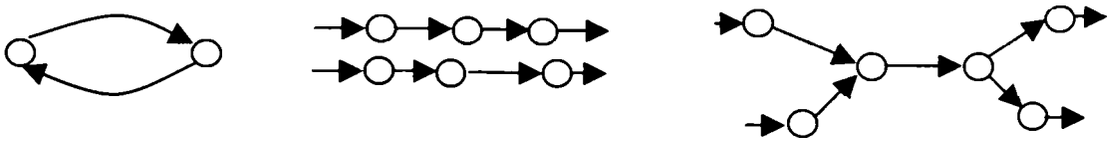
Prin definiţie, *Axioma 3.5* specifică faptul că dacă două puncte de întâlnire sunt separate de o succesiune de intervale atunci există un interval care să conecteze aceste două puncte de întâlnire. Ordonarea a două puncte de întâlnire i şi j se notează $i + j$.

Axioma 3.5

$$\forall i,j \in I : punct(i,j) \Rightarrow \exists k \in I, \ \forall m,n \in I :$$
$$punct(m,i) \wedge punct(j,n) \Rightarrow punct(m,k) \wedge punct(k,n)) \quad (3.5)$$

Axiomele prezentate mai sus nu sunt suficiente pentru a permite extensia sistemului temporal astfel încât să se includă noţiuni de timp paralel şi de ramificare a timpului. Pentru a extinde axiomele prezentate în relaţiile de mai sus se poate consulta lucrarea de referinţă a lui Tsang [125]. De asemenea, nu este clar care dintre cele 5 axiome de mai sus defineşte ipoteza de *liniaritate* a timpului. Deşi autorii indică liniaritatea ca fiind exprimată de *Axioma 3.4*, se poate demonstra că dacă nu ar fi enunţată *Axioma 3.2* atunci timpul în loc să fie *liniar* ar putea fi *circular*, *paralel* sau *ramificat*.

Reprezentarea grafică a timpului în ipotezele circular, paralel sau ramificat se poate exprima ca în *Figura 3.1*:



(a) Timp circular (b) Timp paralel (c) Timp ramificat

Figura 3.1 (a) Timp circular (b) Timp paralel (c) Timp ramificat

În reprezentările din *Figura 3.1*, arcele graficelor corespund intervalelor de timp iar relațiile *punct()* sunt reprezentate ca noduri.

Considerând doar definiția intervalelor ca primitive temporale, deşi punctele sunt introduse în relații sub forma punctelor de întâlnire ale intervalelor, algebra intervalelor nu poate defini relații la nivelul punctelor. După interpretarea lui Galton asupra algebrei intervalelor [56], teoria intervalelor nu este bine adaptată pentru o modelare corectă a evoluției în timp a unui fenomen.

De exemplu, pentru descrierea mişcării unui obiect fizic, folosind algebra temporală, pentru mişcarea unui pendul se obțin două intervale: un interval pentru mişcarea într-o direcție şi un al doilea interval pentru mişcarea în direcția opusă. Legile fizicii evidențiază existența unui punct între aceste două intervale, punct în care obiectul este staționar. Pentru a exprima acest punct staționar în algebra intervalelor avem două posibilități: fie presupunem că există un al treilea interval în care obiectul este staționar, fie presupunem că primul interval se întâlneşte prin primitiva *punct()* cu al doilea interval. Prima ipoteză nu este în concordanță cu legile fizicii, obiectul nu poate fi staționar un interval de timp, ci doar la un moment dat. A doua ipoteză pune problema păstrării unei proprietăți în timpul intervalului. Astfel, algebra intervalelor oferă posibilitatea de a defini o primitivă *menține(proprietate,I)* care înseamnă că o anumită proprietate a obiectului este menținută în timpul intervalului *I*. Pentru cazul prezentat, proprietatea *obiect_în_mişcare* se păstrează în timpul ambelor intervale, într-o direcție sau în cealaltă direcție. Acest fapt se exprimă sub forma următoare:

$$menține(obiect\_în\_mişcare, interval\_1) \quad (3.6)$$

$$menține(obiect\_în\_mişcare, interval\_2) \quad (3.7)$$

Pe durata celor două intervale ar trebui, conform definițiilor algebrei intervalelor, să menținem această proprietate in timpul intervalelor reunite:

$$menține(obiect\_în\_mişcare\ ,\ interval\_1 + interval\_2) \quad (3.8)$$

Relația de mai sus nu poate fi susținută fizic, deoarece la un moment dat în timpul celor două intervale obiectul devine *staționar*, deci proprietatea *obiect_în_mişcare* se pierde în acel punct.

Pentru a putea caracteriza aceste puncte în care anumite evenimente pot interveni, se introduce noțiunea de intervale foarte scurte, numite *momente*. Noțiunea de moment a fost introdusă de către McDermott [83]. Un moment este definit în mod simplu ca fiind un interval de timp care nu mai poate fi descompus în sub-intervale.

Diferența esențială între momente şi puncte este că punctele sunt definite fără nici o limitare între ele iar momentele au un punct de început şi un punct de sfârşit, la fel ca orice

alt interval. Astfel definite, momentele pot întâlni alte intervale, prin intermediul primitivei *punct()*, dar punctele nu pot întâlni nici intervale şi nici alte puncte.

Momentele sunt definite prin următoarea relaţie:

$$\forall m \in I : moment(m) \Leftrightarrow \neg \exists i,j \in I : m = i + j \qquad (3.9)$$

Teoria intervalelor defineşte simultaneitatea a două intervale dacă momentele lor de început şi de sfârşit sunt identice. Pentru a putea asigura această relaţie de simultaneitate, în teoria intervalelor se introduce o nouă axiomă, al cărei scop este de a defini faptul că momentele nu se întâlnesc niciodată:

Axioma 3.6

$$\forall m,n \in I : moment(m) \wedge moment(n) \Rightarrow \neg punct(m,n) \qquad (3.10)$$

Această formulare permite modelări discrete sau continue în timp şi poate permite modelări mai complexe unde există o anumită alternanţă între perioade de timp discrete sau continue. Totuşi, *Axioma 3.6* conduce la o limitare a primitivelor elementelor de timp; un interval este fie imposibil de descompus în sub-intervale, în acest caz constituind un moment, fie este în mod infinit posibil de descompus în sub-intervale. Această limitare se datorează faptului că, dacă un interval ar fi în mod finit posibil de descompus în sub-intervale atunci ar trebui să fie suma unui număr finit de momente, iar în acest caz momentele ar trebui să se întâlnească, ceea ce este contrar în raport cu *Axioma 3.6*. Din acest motiv, algebra intervalelor nu este totuşi indicată pentru modelarea sistemelor discrete.

În cazul în care se doreşte modelarea unor sisteme temporale cu densitate mare, se poate presupune că toate intervalele ar fi în mod infinit posibil de descompus, dar în acest caz nu ar mai exista noţiunile de momente.

Axioma 3.6 introduce noţiunea de momente în algebra intervalelor. În sistemele temporale bazate pe puncte se introduce complet noţiunea de moment.

3.3 Axiomele temporale bazate pe intervale şi puncte

Deoarece algebra intervalelor nu este destul de adaptată pentru a accepta extensii, în modelarea sistemelor discrete [19], [39], se foloseşte sistemul temporal bazat pe intervale şi puncte. Acest sistem temporal este o extensie a algebrei intervalelor prin adăugarea unor axiome pentru definirea punctelor cu ajutorul primitivelor. De asemenea, algebra temporală bazată pe intervale şi puncte permite depăşirea caracteristicii liniare a timpului şi se permite astfel definirea timpului paralel sau ramificat.

Sistemul temporal bazat pe intervale şi puncte introduce noţiunea de mulţime de puncte, notată *T*, care sunt elemente de timp. Se defineşte o funcţie *d* de la *T* la mulţimea numerelor reale pozitive :

$$d : T \rightarrow \Re^{+} \qquad (3.11)$$

Un element de timp *t* este interval dacă $d(t) > 0$ şi este punct în caz contrar. Funcţia *d* mai este denumită şi durata elementelor de timp din mulţimea *T*.

Mulţimea *T* poate fi deci exprimată ca fiind o reuniune a mulţimilor de intervale *I* şi a mulţimilor de puncte *P*:

$$T = I \cup P \qquad (3.12)$$

Pentru a defini primitivele pentru elementele de timp astfel definite, folosim aceeaşi relaţie *punct()* definită în algebra intervalelor, dar trebuie să renunţăm la axioma *Axioma 3.2*, care defineşte ordinea totală a punctelor de întâlnire a intervalelor, deoarece elementele de timp pot fi atât intervale cât şi puncte.

În cadrul teoriei intervalelor şi punctelor, setul de axiome care defineşte elementele de timp pe mulţimea *T* este constituit din axiomele din algebra intervalelor (*Axioma 3.1*, *Axioma 3.3*, *Axioma 3.4* şi *Axioma 3.5*), la care se adaugă axiomele specifice pentru definirea relaţiilor dintre intervale şi puncte (de la *Axioma 3.7* până la *Axioma 3.12*).

Axioma 3.7 este asemănătoare cu *Axioma 3.1* şi precizează că locul unde se întâlnesc două intervale este asociat unui punct:

Axioma 3.7

$$\forall t_1,t_2,t_3,t_4 \in T : punct(t_1,t_2) \wedge punct(t_1,t_3) \wedge punct(t_4,t_2) \Rightarrow punct(t_4,t_3)) \qquad (3.13)$$

Axioma 3.8 este asemănătoare cu *Axioma 3.3* şi precizează că fiecare punct are cel puţin un punct învecinat care îl precede şi un punct învecinat care îl succede:

Axioma 3.8

$$\forall t \in T : \exists t',t'' \in T \text{ a.î. } punct(t',t) \wedge punct(t,t'') \qquad (3.14)$$

Axiomele 3.9 şi *3.10* exprimă relaţiile de ordonare dintre punctele adiacente şi faptul că, într-un interval vor exista întotdeauna puncte:

Axioma 3.9

$\forall t_1,t_2 \in T : \exists t',t'' \in T$ a.î.
$punct(t',t_1) \wedge punct(t_1,t'') \wedge punct(t',t_2) \wedge punct(t_2,t'') \Rightarrow t_1 = t_2$ **(3.15)**

Axioma 3.10

$\forall t_1,t_2 \in T : punct(t_1,t_2) \Rightarrow \exists t \in T$ a.î. $\forall t',t'' \in T :$
$punct(t',t_1) \wedge punct(t_2,t'') \Rightarrow punct(t',t) \wedge punct(t,t'')$ **(3.16)**

Axioma 3.11 exprimă faptul că între două puncte există întotdeauna un interval. *Axioma 3.11* este asemănătoare cu *Axioma 3.6* care specifică faptul că momentele nu pot întâlni alte momente, dar aceasta din urmă conține limitarea că intervalele trebuie să fie în mod infinit posibil de descompus, ceea ce nu este posibil pentru puncte.

Axioma 3.11

$\forall t_1,t_2 \in T : punct(t_1,t_2) \Rightarrow t_1 \in I \vee t_2 \in I$ **(3.17)**

Axioma 3.12 asigură faptul că operația de adăugare a elementelor de timp este consistentă în raport cu funcția *d*, numită funcția de determinare a duratei:

Axioma 3.12

$\forall t_1,t_2 \in T : punct(t_1,t_2) \Rightarrow d(t_1 + t_2) = d(t_1) + d(t_2))$ **(3.18)**

Acest set fundamental de axiome definește complet teoria temporală bazată pe intervale și puncte. Teoria poate fi referită ca un cuplu de elemente *(T,punct)*, unde *T* este mulțimea elementelor de timp iar *punct()* este relația de întâlnire.

3.4 Natura intervalelor

Axiomele descrise în paragraful anterior definesc timpul folosind primitive atât sub formă de intervale cât și sub formă de puncte.

Pentru a obține o definiție completă a sistemului temporal, trebuie detaliate anumite proprietăți complementare ale intervalelor și ale punctelor. Una dintre aceste proprietăți este natura intervalelor, care se clasifică în intervale deschise și intervale închise. Deși nu s-au definit puncte de început sau de sfârșit pentru intervale, o extensie a axiomelor din paragraful precedent poate introduce noțiunile de închis și de deschis pentru intervale.

De exemplu, pentru obiectul în mișcare pe verticală considerat în paragraful anterior, se pot defini proprietățile *obiect_urcare*, *obiect_staționar* și *obiect_coborâre*. Aceste proprietăți sunt asociate intervalului i_1, punctului *p* și respectiv intervalului i_2. Relațiile între cele două intervale și punctul staționar sunt $punct(i_1,p)$ și $punct(p,i_2)$. Pentru a exprima succesiunea *obiect_staționar* + *obiect_coborâre*, putem scrie relația $t = p + i_2$. Deoarece

intervalul t şi punctul p au acelaşi predecesor, acesta fiind intervalul i_1, putem spune că intervalul t este *închis-stânga* în punctul p.

Reprezentarea grafică a intervalului închis-stânga t este cea din *Figura 3.2*:

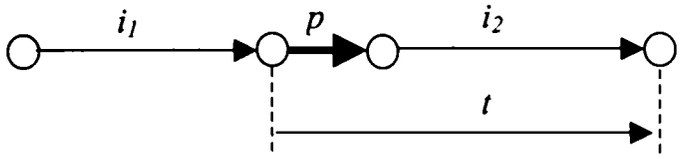


Figura 3.2 Interval *t* închis-stânga

Natura intervalelor deschise şi închise poate fi exprimată formal folosind următoarele definiţii:

Definiţia 3.1

Un interval i este *deschis-stânga* în punctul p dacă şi numai dacă este valabilă expresia:

$$punct(p,i) \qquad (3.19)$$

Definiţia 3.2

Un interval i este *deschis-dreapta* în punctul p dacă şi numai dacă este valabilă expresia:

$$punct(i,p) \qquad (3.20)$$

Definiţia 3.3

Un interval i este *închis-stânga* în punctul p dacă şi numai dacă este valabilă expresia:

$$\exists i' \in I \text{ a.î. } punct(i',i) \wedge punct(i',p) \qquad (3.21)$$

Definiţia 3.4

Un interval i este *închis-dreapta* în punctul p dacă şi numai dacă este valabilă expresia:

$$\exists i' \in I \text{ a.î. } punct(i,i') \wedge punct(p,i') \qquad (3.22)$$

Pornind de la definiţiile de mai sus se observă că intervalele primitive *deschis-stânga* şi *închis-stânga* sunt exclusive între ele. De asemenea, intervalele simetrice *deschis-dreapta* şi *închis-dreapta* sunt exclusive în contextual axiomatic descris de definiţiile de mai sus. Pentru demonstraţie, dacă un interval i este *deschis-stânga* în punctual p_1 şi *închis-stânga* în punctul p_2 atunci, din relaţiile de definiţie ale intervalelor primitive, putem scrie relaţia:

$punct(p_1,i) \wedge punct(i',i) \wedge punct(i',p_2)$ **(3.23)**

Comparând relaţia de mai sus cu *Axioma 3.7*, se determină $punct(p_1,p_2)$, care este contradictoriu cu *Axioma 3.10*, care preciza că două puncte sunt exclusive între ele.

Interpretarea prezentată aici pentru natura intervalelor primitive de a fi deschise sau închise este conformă cu reprezentările convenţionale ale intervalelor bazate pe instanţe temporale. De exemplu, dacă un interval de instanţe $(p_1, p_2]$ este *deschis-stânga* în punctul p_1, deoarece în mod intuitiv p_1 este un predecesor imediat al intervalului $(p_1, p_2]$. În mod similar, $(p_1, p_2]$ este închis-stânga deoarece atât punctul p_2 cât şi intervalul $(p_1, p_2]$ au acelaşi succesor imediat, care este p_2.

Timpul este considerat în general ca având o structură liniară. Această consideraţie corespunde modelului fizic clasic al definiţiei timpului, unde structura folosită este de o linie reală infinită în ambele direcţii.

Liniaritatea completă a unui sistem de elemente temporale *(T,punct)* poate fi caracterizată adăugând o axiomă de liniaritate :

Axioma 3.13

$$\forall t_1,t_2,t_3,t_4 \in T : punct(t_1,t_2) \wedge punct(t_3,t_4) \Rightarrow punct(t_1,t_4)$$
$$\neg\exists t' \in T : punct(t_1,t') \wedge punct(t',t_4)$$
$$\neg\exists t'' \in T : punct(t_3,t'') \wedge punct(t'',t_2) \qquad \textbf{(3.24)}$$

Observăm că *Axioma 3.13* este de fapt a doua axiomă a teoriei intervalelor *Axioma 3.2* descrisă de Allen-Hayes [15]. Operatorul SAU EXCLUSIV în această axiomă implică existenţa unor consecinţe puternice. Astfel, în această axiomă, timpul nu poate fi circular, paralel sau ramificat. Consecinţa axiomei *Axioma 3.13* este următoarea *lemă* (vezi lucrarea [15]), care exprimă imposibilitatea timpului circular :

Lema 3.1

$$\forall t \in T : \neg punct(t,t) \qquad \textbf{(3.25)}$$

Totuşi, fără *Axioma 3.13*, un element temporal permite ramificarea atât în trecut cât şi în viitor. Ramificarea elementelor temporale oferă o posibilitate atractivă de a exprima probabilităţile în trecut şi viitor şi efectele acţiunilor alternative asupra elementelor temporale.

Un element temporal care permite ramificare în viitor dar nu permite ramificare în trecut este numit *liniar-stânga*. Acest element poate fi reprezentat grafic ca în *Figura 3.3*:

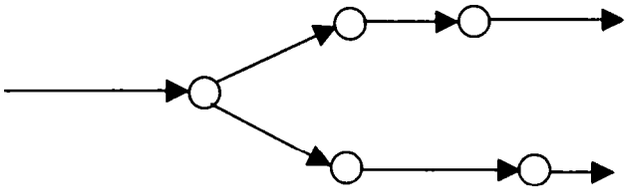


Figura 3.3 Ramificare temporală

Caracterizarea unui element temporal *liniar-stânga* poate fi făcută formal adăugând o axiomă de liniaritate-stânga *Axioma 3.14* în locul axiomei mai puternice *Axioma 3.13*:

Axioma 3.14

$$\forall t_1,t_2,t_3,t_4,t \in T : punct(t_1,t_2) \wedge punct(t_2,t) \wedge punct(t_3,t_4) \wedge punct(t_4,t) \Rightarrow punct(t_1,t_4)$$
$$\neg\exists t' \in T : punct(t_1,t') \wedge punct(t',t_4)$$
$$\neg\exists t'' \in T : punct(t_3,t'') \wedge punct(t'',t_2) \qquad (3.26)$$

În mod similar, se defineşte *Axioma 3.15* pentru a exprima un element temporal *liniar-dreapta*:

Axioma 3.15

$$\forall t,t_1,t_2,t_3,t_4 \in T : punct(t,t_1) \wedge punct(t_1,t_2) \wedge punct(t,t_3) \wedge punct(t_3,t_4) \Rightarrow punct(t_1,t_4)$$
$$\neg\exists t' \in T : punct(t_1,t') \wedge punct(t',t_4)$$
$$\neg\exists t'' \in T : punct(t_3,t'') \wedge punct(t'',t_2) \qquad (3.27)$$

După cum este demonstrat în lucrarea lui Galton [56], este interesant să se observe că *liniaritatea-stânga* şi *liniaritatea-dreapta* nu respectă împreună liniaritatea completă, cu excepţia cazului în care se exprimă timpul paralel, prezentat în *Figura 3.4*:

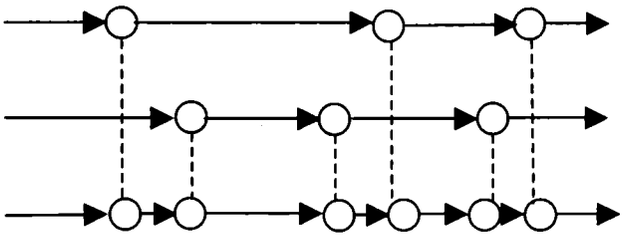


Figura 3.4 Liniaritate completă în timp paralel

Elementele temporale paralele oferă o posibilitate de modelare separată a proceselor asincrone şi pot fi folosite în dezvoltarea logicilor de exprimare în calculul paralele şi procese concurente.

Conform cu *Axioma 3.8*, pentru fiecare element temporal t există un predecesor şi un succesor. *Axioma 3.10* exprimă faptul că între oricare două puncte temporale aflate pe aceeaşi linie temporală, există un interval. Se pune problema dacă orice interval poate fi descompus în două intervale distincte succesive. Dacă această ipoteză este adevărată atunci mulţimea elementelor temporale formează un sistem *dens temporal*.

Pentru a caracteriza densitatea unui element temporal se defineşte următoarea axiomă de densitate, *Axioma 3.16:*

Axioma 3.16

$$\forall i \in I : \exists t_1, t_2 \in T \text{ a.î. } i = t_1 + t_2 \qquad (3.28)$$

Se poate demonstra că *Axioma 3.16* determină faptul că orice interval poate fi descompus în două intervale adiacente. De exemplu, fiind dat intervalul $i = t_1 + t_2$, dacă t_1 este un punct atunci folosind *Axioma 3.10* se determină că t_2 trebuie să fie un interval. Folosind *Axioma 3.16* avem $t_2 = t' + t''$, unde t' şi $t'' \in T$. Din aplicarea pentru *Axioma 3.10* se obţine $i = t_1 + t' + t''$, iar deoarece t_1 este un punct, t' trebuie să fie un interval. Astfel, $i_1 = t_1 + t'$ este un interval şi $i = i_1 + t_2$. În mod similar, se demonstrează că dacă t_2 este un punct atunci t_1 trebuie să fie un interval.

Discretizarea unui sistem complet de elemente temporale *(T,punct)* poate fi caracterizată prin adăugarea a două axiome *discret-stânga Axioma 3.17* şi *discret-dreapta Axioma 3.18:*

Axioma 3.17

$$\forall t \in T \; \exists t_1 \in T : punct(t_1, t) \wedge \neg \exists t_2, t_3 \in T : t_1 = t_2 + t_3 \qquad (3.29)$$

Axioma 3.18

$$\forall t \in T \; \exists t_1 \in T : punct(t, t_1) \wedge \neg \exists t_2, t_3 \in T : t_1 = t_2 + t_3 \qquad (3.30)$$

Axiomele prezentate mai sus introduc noţiunile de *discretizare-stânga* şi de *discretizare-dreapta* a unui sistem temporal. Considerând t drept un interval care nu poate fi descompus, echivalent cu a considera t drept un moment definit în teoria temporală Allen-Hayes, axiomele de discretizare determină faptul că fiecare moment are un moment predecesor şi un moment succesor.

Se poate observa că pot exista sisteme temporale care să nu fie nici discrete şi nici dense. În astfel de sisteme temporale intervalele sunt constituite dintr-o sumă finită de momente, dar în acest caz fiecare interval care poate fi descompus ar trebui să fie în mod

infinit descompus. Se observă că axioma din teoria intervalelor *Axioma 3.6* nu este compatibilă cu axiomele de discretizare.

3.5 Relaţii temporale derivate

Folosind relaţia primitivă *punct()* se poate determina un set complet de relaţii temporale posibile între două elemente temporale, prezentate prin expresiile de mai jos:

- Două elemente temporale pot fi egale:

$$EGAL(t_1,t_2) \Leftrightarrow t_1 = t_2 \qquad (3.31)$$

- Un element temporal poate fi înaintea altui element temporal:

$$ÎNAINTE(t_1,t_2) \Leftrightarrow \exists t \in T : punct(t_1,t) \wedge punct(t,t_2) \qquad (3.32)$$

- Un element temporal poate fi după alt element temporal:

$$DUPĂ(t_1,t_2) \Leftrightarrow ÎNAINTE(t_2,t_1) \qquad (3.33)$$

- Un element temporal poate fi sub un alt element temporal:

$$SUB(t_1,t_2) \Leftrightarrow \exists t,t',t'' \in T : t_1 = t' + t \wedge t_2 = t + t'' \qquad (3.34)$$

- Un element temporal poate fi deasupra unui alt element temporal:

$$DEASUPRA(t_1,t_2) \Leftrightarrow SUB(t_2,t_1) \qquad (3.35)$$

- Un element temporal poate începe un alt element temporal:

$$START(t_1,t_2) \Leftrightarrow \exists t \in T : t_2 = t_1 + t \qquad (3.36)$$

- Un element temporal poate fi început de un alt element temporal:

$$STARTAT(t_1,t_2) \Leftrightarrow START(t_2,t_1) \qquad (3.37)$$

- Un element temporal poate opri un alt element temporal:

$$STOP(t_1,t_2) \Leftrightarrow \exists t \in T : t_2 = t + t_1 \qquad (3.38)$$

- Un element temporal poate fi oprit de un alt element temporal:

$$STOPAT(t_1,t_2) \Leftrightarrow STOP(t_2,t_1) \qquad (3.39)$$

- Un element temporal poate fi conţinut de un alt element temporal:

$$ÎNTRE(t_1,t_2) \Leftrightarrow \exists t' t'' \in T : t_2 = t' + t_1 + t'' \qquad (3.40)$$

- Un element temporal poate conține un alt element temporal:

$CONȚINE(t_1,t_2) \Leftrightarrow ÎNTRE(t_2,t_1)$ **(3.41)**

- Un element temporal poate întâlni un alt element temporal:

$ÎNTÂLNIRE(t_1,t_2) \Leftrightarrow punct(t_1,t_2)$ **(3.42)**

- Un element temporal poate fi întâlnit de un alt element temporal:

$ÎNTÂLNIT(t_1,t_2) \Leftrightarrow INTÂLNIRE(t_2,t_1)$ **(3.43)**

Cele 13 relații prezentate mai sus sunt asemănătoare relațiilor temporale definite între intervale în sistemul temporal propus de Allen [15]. De exemplu, dacă i_1 și i_2 sunt intervale deschise separate de un punct p, atunci se poate scrie $ÎNAINTE(i_1,i_2)$, iar în sistemul temporal al lui Allen se exprimă că i_1 întâlnește pe i_2. În mod similar, dacă i_1 este închis-dreapta și i_2 este închis-stânga, ambele în punctul p, conform definițiilor de mai sus putem exprima $DEASUPRA(i_1,i_2)$ dar în sistemul temporal al lui Allen intervalele se întâlnesc. Se poate exprima și faptul că orice interval deschis este cuprins de un interval închis.

Astfel, considerând atât intervalele cât și punctele ca primitive temporale, trebuie să definim câteva relații suplimentare, deoarece cazul $ÎNTÂLNIRE(i_1,p) \wedge ÎNTÂLNIRE(p,i_2)$ nu este similar cu cazul a două intervale separate de un an 3-lea interval. Aceste 13 relații definite sunt valabile complet în cazul când se consideră numai intervalele ca fiind primitive. Atunci când se consideră și punctele ca fiind primitive, câteva din aceste relații, valabile pentru intervale, nu vor mai fi satisfăcute.

De exemplu, să considerăm punctul $p \in P$. $ÎNTÂLNIRE(p,t_2)$ este valabil și în acest caz, conform cu axiomele punctelor. Dar, considerând cazul următor ***(3.44)***, să analizăm *Axioma 3.11* și *Axioma 3.12 :*

$DEASUPRA(p,t_2) \Leftrightarrow \exists t,t',t'' \in T: p = t' + t \wedge t_2 = t + t''$ **(3.44)**

Axioma 3.12, $d(p) = d(t') + d(t)$ și presupunerea că p este un punct, implică:

$d(t') + d(t) = d(p) = 0$ **(3.45)**

Dar *Axioma 3.11* asigură faptul că cel puțin un interval din t' și t este un interval:

$d(t') + d(t) > 0$ **(3.46)**

Relaţiile de mai sus ***(3.45)*** şi ***(3.46)*** sunt contradictorii şi indică faptul că definiţia *DEASUPRA*(p,t_2) nu poate fi îndeplinită dacă punctele sunt considerate primitive.

Relaţiile posibile între intervale şi puncte pot fi grupate în totalitate astfel:

Relaţii Punct - Punct:

{*EGAL,ÎNAINTE,DUPĂ*}

Relaţii Interval - Interval:

{*EGAL, ÎNAINTE, DUPĂ, DEASUPRA, SUB, START, STARTAT, STOP, STOPAT, ÎNTRE, CONŢINE, ÎNTÂLNIRE, ÎNTÂLNIT*}

Relaţii Punct - Interval:

{*ÎNAINTE, ÎNTÂLNIRE, START, ÎNTRE, STOP, ÎNTÂLNIT, DUPĂ*}

Relaţii Interval - Punct:

{*ÎNAINTE, ÎNTÂLNIRE, STOPAT, CONŢINE, STARTAT, ÎNTÂLNIT, DUPĂ*}

Conform clasificărilor de mai sus, există în total 30 de relaţii temporale posibile asupra elementelor temporale, care pot fi intervale sau puncte. În sistemul temporal prezentat de Vilain [129] sunt indicate 26 dintre aceste 30 de relaţii temporale. Nu sunt indicate relaţiile primitive dintre intervale şi puncte deoarece s-a considerat că primitiva punct este definită doar asupra intervalelor şi nu se aplică interacţiunii dintre intervale şi puncte. Datorită absenţei relaţiilor temporale dintre intervale şi puncte, algebra propusă prezintă anumite dificultăţi pentru modelarea naturii intervalelor (închise sau deschise).

3.6 Modele temporale

Clasificarea modelelor temporale rezultate în urma aplicării teoriilor temporale discutate mai sus poate fi făcută după natura elementelor de timp (*dens* sau *discret*), sau după natura timpului (*liniar*, *paralel* sau *ramificat*).

Pentru a exemplifica un model *dens liniar* vom considera o mulţime de puncte *P* definită peste mulţimea numerelor reale şi o mulţime de intervale *I* construite peste toate perechile de puncte posibile: $p_1,p_2 \in P$ *astfel încât* $p_1 < p_2$.

Folosind notaţiile "<" şi "≤" pentru a exprima relaţiile de ordine peste mulţimea numerelor reale $\Re$, relaţiile temporale sunt următoarele:

$$(p_1,p_2,\text{deschis},\text{deschis}) =_{def} \{ r \in \Re \ / \ p_1 < r < p_2 \} \qquad (3.47)$$

$$(p_1,p_2,\text{deschis},\text{închis}) =_{def} \{ r \in \Re \ / \ p_1 < r \leq p_2 \} \qquad (3.48)$$

$(p_1,p_2,închis,deschis) =_{def} \{ r \in \Re \ / p_1 \leq r < p_2\}$ **(3.49)**

$(p_1,p_2,închis,închis) =_{def} \{ r \in \Re \ / p_1 \leq r \leq p_2\}$ **(3.50)**

Se remarcă faptul că în definiţiile de mai sus intervalele sunt reprezentate de primitive de tipul *interval-stânga* şi *interval-dreapta* care conţin valori din mulţimea de valori $Tip =_{def} \{deschis, închis\}$. Astfel, se pot defini 4 tipuri de intervale bazate pe puncte.

Se face convenţia de a defini *punctul* ca fiind *(p,p,închis,închis)*.

Funcţia *d* (durata) este definită de relaţia următoare:

$d((p_1,p_2,\_,\_)) = p_2 - p_1$ **(3.51)**

Se pot defini în continuare relaţiile *punct()* peste mulţimile $T = P \cup I$:

$punct((p_{11},p_{12},l_1,r_1), (p_{21},p_{22},l_2,r_2)) \Leftrightarrow$ **(3.52)**

$p_{12} = p_{21} \wedge r_1 = deschis \wedge l_2 = închis \vee p_{12} = p_{21} \wedge r_1 = închis \wedge l_2 = deschis$

Acest model satisface axiomele definite pentru puncte, axiomele *3.7, 3.8, 3.9, 3.10, 3.11* şi *3.12* şi, de asemenea, axiomele de liniaritate stânga şi dreapta. Cu alte cuvinte, relaţia de mai sus defineşte modelul temporar liniar şi dens.

Un model discret poate satisface axiomele de puncte *Axioma 3.7* la *Axioma 3.12*, *discret-stânga Axioma 3.17* şi *discret-dreapta Axioma 3.18* şi poate fi construit prin considerarea limitării punctelor la mulţimea numerelor întregi. Se poate observa că, în cazul modelelor discrete, deşi punctele nu se întâlnesc niciodată între ele, intervalele nu sunt în mod infinit posibil de descompus.

De exemplu, pornind de la definiţia de mai sus, intervalul *(1,3,deschis,închis)* poate fi descompus în maxim 4 elemente de timp care la rândul lor nu mai pot fi descompuse:

(1,3,dechis,închis)=(1,2,deschis,deschis)+(2,2,închis,închis)+(2,3,deschis,deschis)+ (3,3,închis,închis) **(3.53)**

Acest model nu este valid pentru *Axioma 3.6* a teoriei intervalelor a lui Allen-Hayes, care presupune că dacă un interval este posibil de descompus atunci el trebuie să fie în mod infinit posibil de descompus. Altfel, dacă un interval este finit posibil de descompus atunci el trebuie să fie suma unui număr finit de momente care se întâlnesc.

Pentru a interpreta corect axiomele lui Allen-Hayes, trebuie exclusă axioma *Axioma 3.6* pentru că nu este consistentă cu timpul discret. În schimb *Axioma 3.5* poate fi satisfăcută de aceste modele discrete.

Axiomele prezentate până aici sunt un element de definiţie suficient pentru a descrie mai multe sisteme temporale, cum sunt teoriile bazate pe puncte ale lui Bruce [28], McDermott [83] şi teoriile bazate pe intervale ale lui Allen, Galton, precum şi teoriile mixte cu puncte şi intervale ale lui Vilain [129] şi Knight-Ma [71].

3.6.1 Teoria temporală Bruce bazată pe puncte

Teoria temporală Bruce bazată pe puncte este definită de o mulţime simplă de puncte care au o ordine parţială. În teorie se notează relaţia de ordin parţial $(\trianglelefteq)$ asupra mulţimii de puncte P:

$$p_1 \trianglelefteq p_2 \Leftrightarrow EGAL(p_1,p_2) \vee ÎNAINTE(p_1,p_2), \qquad \textbf{(3.54)}$$

unde *EGAL* şi *ÎNAINTE* sunt relaţii introduse în paragraful anterior §3.5. Sistemul temporal al lui Bruce este deci notat astfel: $(P,\trianglelefteq)$ definit pe *(T,punct)*

În mod similar se pot defini cele 7 relaţii binare asupra managementului timpului [28]. De remarcat că teoriile temporale ale lui Ladkin [74], Dechter [40] şi Maiocchi [79] sunt similare cu teoria lui Bruce în sensul că toate intervalele sunt construite din puncte.

3.6.2 Logica temporală McDermott

Modelul temporal propus de McDermott în lucrările [83], [84] şi [85], dezvoltă o logică bine adaptată modelării evoluţiei în timp a unor fenomene. Teoria temporală presupune o relaţie de tipul *cel târziu*, definită peste o colecţie de puncte dense şi care definesc o linie temporală liniară la stânga şi ramificată în viitor. Astfel, în viitor există mai multe ramuri posibile pentru evoluţia sistemului. Fiecare ramură temporală din viitor constituie o *cronică* şi este compusă dintr-o mulţime densă de puncte [16].

Considerând axiomele de definiţie *Axioma 3.7* la *Axioma 3.12* şi liniaritate-stânga *Axioma 3.14* şi mai trebuie definită o axiomă adiţională de punct-dens *Axioma 3.19* pentru a defini că există întotdeauna un punct în timpul unui interval:

Axioma 3.19

$$\forall i \in I : \exists p \in P, \exists i_1,i_2 \in I \text{ a.î. } i = i_1 + p + i_2 \qquad \textbf{(3.55)}$$

Dacă se consideră *Axioma 3.8* şi *Axioma 3.11* se poate determina că *Axioma 3.19* exprimă faptul că între două puncte distincte pe aceeaşi linie temporală există un al treilea punct. De fapt, *Axioma 3.19* este mai puternică decât axioma de densitate *Axioma 3.16*

La fel ca şi în teoria lui Bruce, putem să definim relaţiile de tipul *nu mai târziu* de asupra punctelor, respectând condiţiile *EGAL* şi *ÎNAINTE*. Astfel, se poate considera structura timpului propusă de McDermott [83] ca un model care accesează doar punctele şi o relaţie de tipul *nu mai târziu*.

3.6.3 Teoria intervalelor Allen-Hayes

Axiomele intervalelor prezentate aici sunt extensii ale teoriei intervalelor Allen-Hayes [15], care defineşte intervalele ca primitive şi tratează punctele în formă abstractă. Relaţia importantă definită pentru intervale este *CONŢINUT*, definită astfel:

$$CONŢINUT(p, i) \Leftrightarrow \forall i' \in I : ÎN(i', i) \Rightarrow CONŢINUT(p, i') \qquad (3.56)$$

În ecuaţia **(3.56)** de mai sus apare relaţia *ÎN*, care este definită în raport cu relaţiile temporale astfel:

$$ÎN(i', i) \Leftrightarrow ÎNTRE(i', i) \wedge START(i', i) \vee STOP(i', i) \qquad (3.57)$$

Negarea relaţiei *CONŢINUT* se exprimă astfel:

$$CONŢINUT(\neg pct, i) \Leftrightarrow \forall i' \in I : ÎN(i', i) \Rightarrow \neg CONŢINUT(pct, i')) \qquad (3.58)$$

Problema care poate apărea în teoria intervalelor este distingerea între două tipuri de proprietăţi: poziţie şi mişcare. Poziţia izolează puncte, iar mişcarea nu poate izola puncte. Pentru detalii asupra acestei probleme poate fi consultată lucrarea [55].

3.6.4 Sistemul temporal bazat pe intervale şi puncte Vilain

Deoarece intervalele nu sunt singura reprezentare a timpului, un sistem temporal poate fi descris de puncte, cum este teoria temporală propusă de Vilain [129], [130].

Acest sistem temporal extinde cele 13 relaţii temporale din teoria intervalelor pentru a obţine 26 de relaţii între puncte şi intervale. Menţionăm că cele 26 de relaţii constituie o submulţime a celor 30 de relaţii care le-am prezentat în paragraful anterior.

Relaţiile care nu sunt incluse în sistemul temporal propus de Vilain sunt următoarele: *ÎNTÂLNIRE, ÎNTÂLNIT* între puncte şi intervale şi *ÎNTÂLNIRE, ÎNTÂLNIT* între puncte şi intervale.

Teoria lui Vilain este mai restrictivă deoarece conţine o axiomă care defineşte că dacă două elemente temporale se întâlnesc atunci amândouă trebuie să fie intervale:

Axioma 3.20

$$\forall t_1,t_2 \in T : punct(t_1,t_2) \Rightarrow t_1 \in I \wedge t_2 \in I \qquad (3.59)$$

3.6.5 Modelul temporal Knight-Ma

În lucrarea [71], Knight şi Ma introduc un model temporal care consideră atât punctele cât şi intervalele ca fiind primitive temporale. Acest model temporal este adaptat mulţimilor finite de elemente temporale.

Sistemul temporal Knight-Ma este bazat pe definirea unor elemente temporale fundamentale asemănătoare noţiunii de momente, aşa cum a fost definită în sistemul temporal Allen-Hayes.

Menţionăm că, pentru modelarea unei baze de date în informatică, se utilizează doar un număr finit de elemente, astfel definind un număr dens şi discret de elemente primitive.

3.7 Concluzii

Teoriile temporale generale, prezentate în acest capitol, permit definirea problemei de reprezentare temporală în scopul alegerii metodei potrivite de modelare matematică.

Am prezentat diferitele axiome temporale descrise în lucrările referitoare la teoriile temporale. Între aceste axiome putem distinge axiomele de liniaritate şi cele de densitate, care sunt cele mai apropiate de modelarea generării unor evenimente discrete, cum sunt alarmele în cadrul reţelelor de telecomunicaţii.

Pentru descrierea problemei de recunoaştere a cronicilor de alarme am ales modelul temporal propus de McDermott [83], definit peste o colecţie de puncte dense cu o linie temporală liniară la stânga şi ramificată în viitor. Linia temporală liniară la stânga este potrivită pentru reprezentarea succesiunilor de alarme, iar în viitor există mai multe ramuri posibile pentru evoluţia sistemului. Fiecare ramură temporală din viitor constituie o posibilă *cronică* şi este compusă dintr-o mulţime densă de puncte, care sunt *alarme* în cazul reţelelor de telecomunicaţii.

4.1 Definiţii ale reţelelor Petri

În domeniul modelării matematice, reţelele Petri sunt recunoscute pentru faptul că exprimă foarte bine fenomenele de concurenţă şi de cauzalitate ale unor evenimente. Astfel, reţelele Petri permit o exprimare grafică simplă şi intuitivă pentru descrierea acestor fenomene. Concurenţa unor evenimente se referă la producerea lor într-o anumită succesiune iar cauzalitatea se referă la relaţiile dintre aceste evenimente [69], [104].

Reţelele Petri au fost introduse pentru prima dată în anul 1962, în teza de doctorat *"Kommunikation mit Automaten"* a lui Carl Adam Petri [100], susţinută la Universitatea Tehnică din Darmstadt, Germania. Pentru o introducere mai aprofundată în formalismul reţelelor Petri pot fi consultate lucrările de referinţă ale lui Murata [87] şi Peterson [99].

Definiţia unei reţele Petri, prezentată în forma simplificată, este următoarea:

Definiţia 4.1

O reţea Petri, notată N=(P,T,L), este un graf orientat bipartit, definit pe două mulţimi finite şi distincte de elemente: mulţimea poziţiilor P şi mulţimea tranziţiilor T. Legăturile sau arcele dintre aceste elemente, notate L, conectează poziţiile la tranziţii sau tranziţiile la poziţii. Legăturile satisfac condiţia:

$$L \subset (P \times T) \cup (T \times P) \qquad (4.1)$$

Condiţia exprimată prin relaţia ***(4.1)*** se interpretează intuitiv prin expresia legăturilor ca fiind incluse în spaţiul reunit al mulţimilor poziţiilor şi al tranziţiilor.

În mod tradiţional, simbolizarea grafică a poziţiilor se reprezintă prin cercuri, iar tranziţiile se reprezintă prin bare sau dreptunghiuri. Legăturile pleacă fie de la o poziţie la o tranziţie, fie de la o tranziţie la o poziţie. Nu există legături care să conecteze două poziţii între ele, după cum nu există legături care să conecteze două tranziţii între ele. De asemenea, în forma simplă a reţelelor Petri, presupunem că nu există nici poziţii şi nici tranziţii neconectate de o legătură, deci nu există poziţii izolate şi nici tranziţii izolate.

În general, poziţiile reprezintă resurse sau condiţii logice, iar tranziţiile reprezintă acţiuni sau evenimente.

Legăturile sunt etichetate cu ponderile lor, valori întregi şi pozitive. O legătură cu ponderea k poate fi privită ca o mulţime de k legături paralele cu pondere unitară. În general, în reprezentările grafice uzuale, nu este necesar să fie indicate etichetele pentru ponderea unitară.

Presetul unei tranziţii dintr-o reţea Petri se defineşte astfel:

Definiţia 4.2

Pentru toate tranziţiile $t \in T$, se defineşte presetul $\bullet t$, ca fiind mulţimea tuturor poziţiilor conectate la t în amonte, adică:

$$\bullet t = \{p \in P : (p,t) \in L\} \qquad (4.2)$$

Poziţiile din presetul $\bullet t$, numite şi locuri de intrare pentru t, reprezintă precondiţiile, resursele necesare sau datele de intrare pentru evenimentul t.

În mod similar, se defineşte postsetul unei tranziţii dintr-o reţea Petri astfel:

Definiţia 4.3

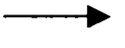
Pentru oricare tranziţie $t \in T$, se defineşte postsetul $t \bullet$, ca fiind mulţimea tuturor poziţiilor conectate la t în aval, adică:

$$t\bullet = \{p \in P : (t,p) \in L\} \qquad (4.3)$$

Poziţiile din mulţimea $t \bullet$, numite şi locuri de ieşire pentru t, exprimă în general postcondiţiile, resursele create sau datele de ieşire corespunzătoare evenimentului t.

Dinamica reţelelor Petri este realizată prin introducerea noţiunii de *jeton* ca fiind un element asociat unei poziţii şi care poate fi deplasat în reţeaua Petri în urma execuţiei unei tranziţii.

Pentru reprezentarea grafică a reţelelor Petri se folosesc următoarele simboluri grafice :

- Poziţie p : 
- Marcajul unei poziţii p cu un jeton : 
- Tranziţie t : 
- Legătură între poziţie şi tranziţie (sau invers) : 

Definiţia unei bucle între o poziţie p şi o tranziţie t este următoarea:

Definiţia 4.4

O poziţie p formează o buclă în raport cu tranziţia t dacă p se află în acelaşi timp în presetul şi în postsetul acelei tranziţii, ceea ce se exprimă prin relaţia:

$$p \in {}^\bullet t \cap t^\bullet \qquad (4.4)$$

Un exemplu de reprezentare grafică a unei bucle exprimate de relaţia ***(4.4)*** este dat în *Figura 4.1* :

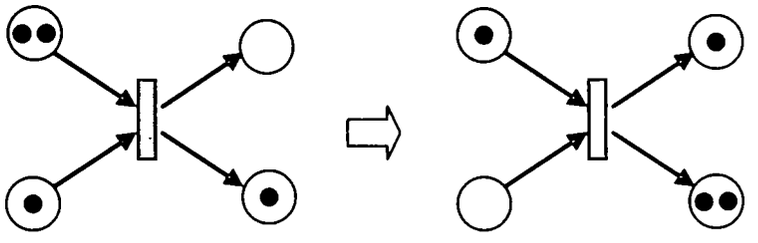


Figura 4.1 Exemple de preset şi postset ale unei tranziţii

În *Figura 4.1* se observă că trecerea este autorizată, deoarece în partea stângă există preseturi care conţin cel puţin un jeton. În partea dreaptă, se prezintă rezultatul tranziţiei: în preset s-a redus numărul jetoanelor şi în postset a crescut numărul jetoanelor. În acest exemplu există o buclă între poziţii şi tranziţia reţelei Petri.

Atunci când o reţea Petri nu conţine nici o buclă între poziţii şi tranziţii se defineşte noţiunea de reţea Petri pură:

Definiţia 4.5

O reţea Petri pură este o reţea Petri care nu conţine nici o buclă între poziţii şi tranziţii.

În mod similar cu preseturile şi postseturile tranziţiilor $t \in T$ se pot defini preseturile şi postseturile poziţiilor $p \in P$, folosind relaţiile următoare :

$$ {}^\bullet p = \{t \in T : (t, p) \in L\} \qquad (4.5)$$

$$ p^\bullet = \{t \in T : (p, t) \in L\}. \qquad (4.6)$$

Marcajul unei reţele Petri se defineşte ca fiind o funcţie $M : P \mapsto \mathrm{N}$, unde N este mulţimea numerelor naturale. Reprezentarea grafică a unui marcaj este o distribuţie de jetoane în poziţiile reţelelor Petri.

Un marcaj M este denumit k-bornat dacă în toate poziţiile $p \in P$ numărul de jetoane $M(p)$ este mai mic sau egal decât k. Un marcaj iniţial M_0 poate fi asociat reţelelor Petri $N = (P,T,L)$. În acest caz, reţelele Petri care conţin un marcaj iniţial se numesc reţele Petri marcate şi se notează sub forma (N,M_0) sau (P,T,L,M_0).

Marcajul reprezintă starea globală a sistemului. Dinamica reţelelor Petri este exprimată prin validarea şi trecerea unor tranziţii, fapt care schimbă marcajul reţelei Petri. Pentru ca o tranziţie să fie trecută, ea trebuie să fie autorizată în marcajul curent. Validarea unei tranziţii presupune că trebuie îndeplinită condiţia ca toate poziţiile de preset să conţină cel puţin un jeton.

Definiţia 4.6

Considerând o tranziţie $t \in T$ autorizată într-un marcaj iniţial M_0, trecerea către marcajul M_1 se exprimă prin relaţia:

$$M_1(p) = M_1(p) + I_{\{p \in t^\bullet\}} - I_{\{p \in {}^\bullet t\}}, \forall p \in P \qquad (4.7)$$

Trecerea din marcajul iniţial M_0 către marcajul M_1 se notează astfel :

$$M_0 [t\rangle M_1 \qquad (4.8)$$

Explicaţia intuitivă pentru *Figura 4.1* este că la trecerea unei tranziţii se extrage un jeton de la fiecare poziţie din preset şi se adaugă acel jeton la poziţiile din postset. Astfel, dacă jetoanele sunt privite ca resurse, se poate spune că tranziţiile consumă resursele poziţiilor de intrare şi creează resurse în poziţiile de ieşire.

O remarcă importantă este faptul că o resursă nu poate fi creată sau consumată decât de către o tranziţie. În acest mod, trecerile de tranziţii conduc la propagarea jetoanelor şi definesc dinamica reţelelor Petri. O tranziţie poate fi trecută sau efectuată, dar nu poate fi forţată. De asemenea, că regulile de trecere sau de autorizare a tranziţiilor sunt locale şi nu depind decât de poziţiile din vecinătatea tranziţiei. De aceea, două tranziţii autorizate, care nu consumă aceleaşi resurse, ar putea în principiu să fie trecute simultan. Aceste tranziţii sunt numite tranziţii concurente.

Definiţia 4.7

O secvenţă executabilă (s), numită şi execuţie a unei reţele Petri, pornind de la marcajul iniţial M_0, este o secvenţă de treceri de tranziţii ($s = t_1 ... t_n$), astfel încât:

$$M_0 [t_1\rangle M_1 ... M_{n-1} [t_n\rangle M_n \qquad (4.9)$$

Secvenţele de execuţie a unei reţele Petri se notează astfel :

$$M_0 [s\rangle M_n \qquad (4.10)$$

O secvenţă este executabilă dacă şi numai dacă există un marcaj iniţial pornind de la care secvenţa este executabilă.

Un marcaj M_n se numeşte accesibil de la marcajul iniţial M_0, dacă există cel puţin o secvenţă executabilă de la M_0 la M_n, conform relaţiei ***(4.10)***.

Ansamblul marcajelor accesibile pornind de la M_0 se notează $\mu(M_0)$. Regulile de trecere definesc un graf pe ansamblul de marcaje accesibile.

Definiţia 4.8

Graful marcajelor unei reţele Petri $N = (P, T, L, M_0)$ este un graf orientat (R, G), unde mulţimea nodurilor este mulţimea marcajelor accesibile $R = \mu(M_0)$, iar mulţimea legăturilor $G \subset R \times T \times R$ este definită de:

$$(M, t, M') \in G \leftrightarrow M [t\rangle M' \qquad (4.11)$$

Graful marcajelor reprezintă spaţiul poziţiilor reţelei Petri, unde schimbările de stări corespund trecerilor tranziţiilor şi pot fi considerate ca evenimente discrete. Altfel spus, graful marcajelor este un automat asociat reţelelor Petri.

Se observă că în general mulţimea $\mu(M_0)$, care defineşte nodurile graficului de marcaj, poate fi o mulţime infinită. Reţelele Petri care nu o mulţime infinită de noduri se numesc reţele Petri bornate.

Definiţia 4.9

Fie k un număr întreg. Reţelele Petri (N, M_0) se numesc k-bornate dacă toate marcajele accesibile pornind de la M_0 sunt k-bornate.

Pentru rețelele Petri k-bornate, mulțimea marcajelor accesibile $\mu(M_0)$ este finită și conține maxim $(k+1)^{|P|}$ elemente, unde $|P|$ este numărul de poziții în N.

Rețelele *1*-bornate sunt denumite rețele Petri reduse. În mod similar, marcajele pot fi *1*-bornate. După cum este demonstrat în lucrarea [87], toate rețelele Petri bornate pot fi simplificate și transformate în rețele Petri reduse, care sunt echivalente și generează același comportament ca și rețeaua inițială. Altfel spus, rețelele Petri reduse au aceeași putere de modelare ca și rețelele Petri bornate.

Un marcaj sau o stare atribuie fiecărei poziții un număr întreg pozitiv. Dacă un marcaj atribuie poziției p un număr întreg $k > 0$, se spune că p este marcat cu k jetoane. Din punct de vedere grafic, în cercul corespunzător poziției p se vor plasa k obiecte care simbolizează jetoanele.

Orice marcaj M_i este un vector m-dimensional, unde m notează numărul total al pozițiilor. Componenta p a lui M_i, notată $M_i(p)$, reprezintă numărul de jetoane din poziția p.

În problemele de modelare care utilizează conceptele de condiții și evenimente, pozițiile reprezintă condiții și tranzițiile reprezintă evenimente. O tranziție (eveniment) posedă un număr de poziții de intrare și ieșire, care reprezintă pre-condiții și respectiv post-condiții pentru evenimentul în cauză. Prezența unui jeton într-o poziție trebuie înțeleasă ca valoare logică pentru condiția asociată respectivei poziții (condiție adevărată).

Aspectele de ponderare a arcelor și de marcaj inițial, prezentate anterior, exprimă complet o extindere a noțiunii de rețea Petri prin următoarea definiție:

Definiția 4.10

O rețea Petri este un cvintuplu, $PN = (P, T, F, W, M_0)$ *pentru care:*

- $P = \{p_1, p_2, ..., p_n\}$ *este o mulțime finită de poziții,*
- $T = \{t_1, t_2, ..., t_n\}$ *este o mulțime finită de tranziții,*
- $F \subseteq (P \times T) \cup (T \times P)$ *este o mulțime de arce sau legături,*
- $W : F \to \{1,2,...,n\}$ *este o funcție de ponderare a arcelor,*
- $M_0 : P \to \{0,1,2,...,n-1\}$ *este o funcție de marcaj inițial.*

Pentru a aprofunda formalismul rețelelor Petri este necesară exprimarea unor alte noțiuni referitoare la rețele Petri, prezentate în continuare:

- Mulțimile P și T sunt disjuncte: $P \cap T = \emptyset$,

- Pentru a asigura obiectul definiției de mai sus, mulțimile P și T satisfac condiția $P \cup T \neq \emptyset$,
- Structura de rețea Petri $N = (P, T, F, W)$ fără nici o specificație referitoare la marcaj se notează N, notație care desemnează topologia rețelei,
- Rețeaua Petri cu un marcaj inițial M_0 se notează (N, M_0),
- Rețeaua Petri cu un marcaj oarecare M_i se notează (N, M_i).

4.2 Validarea și executarea tranzițiilor (dinamica rețelelor Petri)

Marcajul unei rețele Petri are semnificația de stare a rețelei și se poate modifica în conformitate cu următoarele două definiții, denumite regula tranziției de validare și regula tranziției de executare :

Definiția 4.11

O tranziție t se spune că este validată dacă fiecare poziție de intrare p a lui t este marcată cu cel puțin W(p,t) jetoane, unde W(p,t) notează ponderea arcului de la p la t.

Definiția 4.12

O tranziție validată poate să fie executată sau să nu fie executată, după cum evenimentul asociat tranziției are loc sau nu are loc.

Executarea unei tranziții validate *t* elimină *W(p, t)* jetoane din fiecare poziție de intrare *p* a lui *t* și adaugă *W(t, p)* jetoane la fiecare poziție de ieșire *p* a lui *t*, unde *W(t, p)* este ponderea arcului de la *t* la *p*. O rețea Petri se numește ordinară dacă toate arcele sale au pondere unitară.

O tranziție fără nici o poziție de intrare se numește tranziție *sursă*. O tranziție fără nici o poziție de ieșire se numește tranziție *receptor*. Modul de operare al acestor tranziții este următorul:

- O tranziție sursă este necondiționat validată (fără a fi obligatoriu să se execute), iar executarea ei produce jetoane.
- O tranziție receptor consumă jetoane atunci când este executată.

Pe întreg parcursul modelării cu rețele Petri vom considera că executarea unei tranziții nu consumă timp și că jetoanele pot rămâne în pozițiile lor respective o durată de timp indefinită. Astfel, deoarece executarea unei tranziții este instantanee, se consideră că tranzițiile se execută numai secvențial și nu pot fi executate simultan.

Folosind ac estor ipoteze de lucru cu rețele Petri vom putea alege rețelele Petri pentru a modela relațiile de concurență și de cauzalitate între anumite succesiuni de evenimente.

4.3 Rețele Petri de capacitate finită și infinită

Pentru regula de validare a unei tranziții care a fost prezentată anterior, s-a presupus că fiecare poziție poate conține un număr nelimitat de jetoane. O astfel de rețea se numește rețea Petri de capacitate infinită.

În modelarea sistemelor fizice este firesc să se considere o limită superioară a numărului de jetoane pe care îl poate conține fiecare poziție, considerând doar rețele Petri de capacitate finită.

Într-o rețea Petri de capacitate finită *(N, M0)*, fiecărei poziții *p* i se asociază capacitatea poziției, notată *K(p)*, definită ca numărul maxim de jetoane ce pot fi conținute în *p*. Pentru validarea unei tranziții *t* este necesară următoarea condiție suplimentară: numărul de jetoane în fiecare poziție de ieșire *p* a lui *t* nu poate să depășească capacitatea poziției respective, *K(p)*, a tunci c ând *t* se execută. Cu această condiție de capacitate a poziției, regula tranziției din *Definiția 4.11* se va numi regula *strictă* a tranziției.

Fiind dată o rețea Petri de capacitate finită (N, M_0) este posibil să se aplice regula strictă a tranziției direct pentru rețeaua (N, M_0) sau se poate aplica regula simplă a tranziției pentru o rețea transformată în prealabil, notată (N', M'_0). Presupunând că rețeaua *N* este pură, următorul algoritm permite construcția rețelei (N', M'_0) pornind de la rețeaua (N, M_0), prin metoda pozițiilor complementare :

- *Pasul i.* Pentru fiecare poziție *p*, se adaugă o poziție complementară *p'*, al cărei marcaj inițial este dat de $M'_0(p') = K(p) - M_0(p)$.
- *Pasul i+1.* Între fiecare tranziție *t* și unele poziții complementare *p'* se trasează arce suplimentare, *(t,p')* sau *(p',t)*, cu ponderile $W(t,p')=W(p,t)$, respectiv $W(p',t)=W(t,p)$, astfel încât suma jetoanelor în poziția *p* și în poziția complementară corespunzătoare *p'* să fie egală cu capacitatea *K(p)*, atât înainte, cât și după executarea tranziției *t*. În acest mod se asigură satisfacerea condiției $M(p)+M'(p')=K(p)$.

Se observă faptul că algoritmul prezentat mai sus nu adaugă tranziții suplimentare în rețeaua Petri transformată.

Teorema 4.1

Fie (N, M_0) o rețea pură, cu capacitate finită, căreia i se aplică regula strictă a tranziției. Fie (N', M'_0) rețeaua obținută prin transformarea lui (N, M_0), utilizând metoda pozițiilor complementare. În (N', M'_0) se aplică regula simplă a tranziției. Rețelele (N,

M_0) şi (N', M'_0) sunt echivalente în sensul că ambele posedă aceeaşi mulţime de secvenţe posibile pentru executarea tranziţiilor.

În baza teoremei de mai sus, se constată că regula strictă a tranziţiei într-o reţea cu capacitate finită poate fi întotdeauna înlocuită de mai multe reguli simple ale tranziţiilor echivalente într-o reţea cu capacitate infinită. Din acest motiv, pentru modelarea cu reţele Petri se preferă utilizarea reţelelor cu capacitate infinită şi utilizarea regulii simple a tranziţiei în aceste reţele. Acest mod de reprezentare este, de fapt, cel mai des întâlnit în lucrările de specialitate.

4.4 Proprietăţi ale reţelelor Petri

Facilităţile de modelare oferite de reţelele Peri se datorează proprietăţilor lor, care sunt prezentate în continuare. Noţiunile prezentate în acest paragraf pot fi regăsite în referinţele bibliografice [29] şi [31].

4.4.1 Conflict, decizie sau alegere liberă

Două evenimente e_1 şi e_2 sunt în conflict dacă pot apărea amândouă separat dar nu simultan. În *Figura 4.2* se prezintă o structură care modelează conflictul dintre evenimentele e_1 şi e_2, materializate prin tranziţiile t_1, respectiv t_2. Ambele tranziţii sunt validate dar numai una dintre ele se poate executa, execuţia ei conducând la invalidarea celeilalte tranziţii.

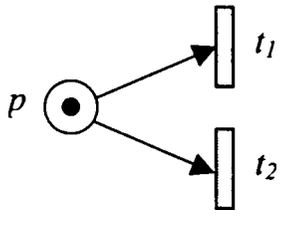


Figura 4.2 Conflict, decizie sau alegere liberă

În funcţie de natura aplicaţiei, structura din *Figura 4.2* poate modela procesul de luare a unei decizii sau procesul de alegere liberă. Din acest motiv structura se numeşte *conflict* sau *decizie* sau *alegere liberă.*

4.4.2 Alegere liberă extinsă

În raport cu alegerea liberă, în *Figura 4.3* ambele tranziţii t_1 şi t_2 sunt validate, de asemenea, dar numai una dintre ele se va executa, conducând la invalidarea celeilalte.

Executarea uneia şi numai a uneia dintre cele două tranziţii va consuma ambele jetoanele din p_1 şi p_2.

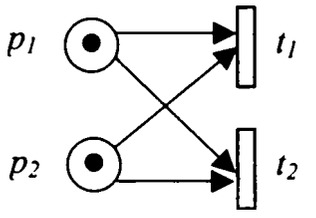


Figura 4.3 Alegere liberă extinsă

Deoarece aici trebuie făcută o alegere privind executarea fie a tranziţiei t_1, fie a tranziţiei t_2, structura din *Figura 4.3* este denumită *alegere liberă extinsă*, în raport cu *alegerea liberă* prezentată anterior în *Figura 4.2*.

4.4.3 Alegere asimetrică

În *Figura 4.4* ambele tranziţii t_1 şi t_2 sunt validate, de asemenea, dar numai una dintre ele se va executa, conducând la invalidarea celeilalte. Deoarece aici trebuie făcută a alegere asimetrică privind executarea lui t_1 sau t_2, structura din *Figura 4.4* este denumită *alegere asimetrică*, pentru a o deosebi de *alegerea liberă* prezentată în *Figura 4.2* şi de *alegerea liberă extinsă* prezentată în *Figura 4.3*:

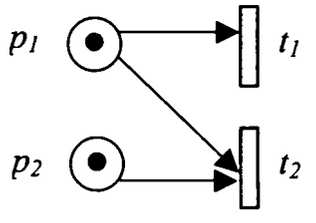


Figura 4.4 Alegere asimetrică

4.4.4 Paralelism sau concurenţă

În *Figura 4.5*, ambele poziţii p_1 şi p_2 primesc câte un jeton în urma executării tranziţiei t. Astfel, aceste poziţii pot modela două activităţi care se desfăşoară în paralel. Tranziţiile t_1 şi t_2 se pot executa independent una faţă de cealaltă, indiferent de ordine, modelând astfel două evenimente concurente, care apar fără a genera un conflict. Structura din *Figura 4.5* se numeşte *paralelism* sau *concurenţă*.

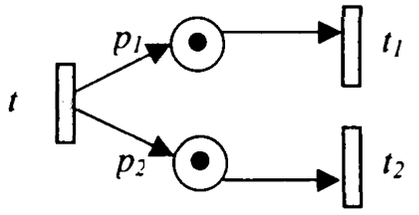


Figura 4.5 Paralelism sau concurenţă

4.4.5 Confuzie

Situaţiile caracterizate atât prin concurenţă cât şi prin conflict sunt denumite *confuzii*. În *Figura 4.6* se prezintă o structură denumită *confuzie simetrică*: evenimentele corespunzătoare lui t_1 şi t_2 sunt concurente, în timp ce fiecare din aceste evenimente este în conflict cu evenimentul corespunzător lui t_3.

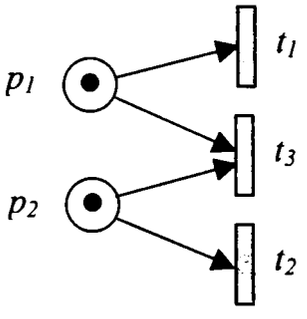


Figura 4.6 Confuzie simetrică

În *Figura 4.7* se prezintă o structură denumită *confuzie asimetrică*: t_1 este concurent cu t_2, dar t_1 va fi în conflict cu t_3, dacă t_2 se execută înaintea lui t_1.

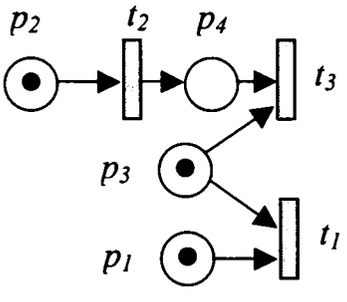


Figura 4.7 Confuzie asimetrică

4.4.6 Sincronizare

În *Figura 4.8* este prezentată structura de *sincronizare*: poziţiile p_1 şi p_2 reprezintă două activităţi care se pot desfăşura în paralel şi a căror încheiere trebuie corelată sau sincronizată. Prin executarea tranziţiei t se extrage câte un jeton atât din p_1 cât şi din p_2.

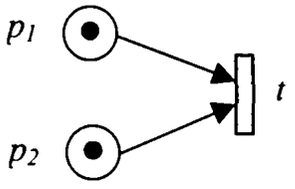


Figura 4.8 Sincronizare

4.4.7 Post-condiţie comună

În *Figura 4.9* este prezentată structura de *post-condiţie comună*: tranziţiilor t_1 şi t_2 le sunt asociate două evenimente care se pot produce independent, apariţia oricăruia dintre ele conducând la satisfacerea condiţiei asociate poziţiei p. Prin executarea oricăreia din tranziţiile t_1 sau t_2 se introduce un jeton în p.

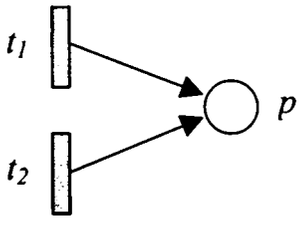


Figura 4.9 Post-condiţie comună

4.5 Studiu comparativ între automate şi reţele Petri

În acest paragraf se prezintă un studiu privind compararea modelelor bazate pe automate faţă de modelele bazate pe reţele Petri, din punct de vedere al analizei unor procese pilotate de evenimente discrete.

Obiectivul principal al acestui paragraf este de a prezenta motivele pentru care am ales modelarea cu reţele Petri, faţă de modelarea cu automate, pentru a analiza cronicile de alarme de telecomunicaţii.

4.5.1 Automate pentru modelarea sistemelor cu evenimente discrete

Automatele, definite mai jos, sunt, de asemenea, mijloacele potrivite pentru modelarea comportamentului unor sisteme dinamice bazate pe evenimente discrete.

Definiția 4.13

Un automat este un cvintuplu de elemente (E, X, P, f, x_0), pentru care semnificația obiectelor matematice este următoarea:

- *E este mulțimea evenimentelor (fiind o mulțime numărabilă de elemente $e \in E$),*
- *X este spațiul stărilor (fiind o mulțime numărabilă de elemente $x \in X$),*
- *P este mulțimea evenimentelor posibile (care se pot produce) în starea $x \in X$,*
- *f este funcția de tranziție a stărilor: $f : X \times E \mapsto X, x' = f(x,e)$,*
- *x_0 este starea inițială.*

Funcția f este definită numai pentru $e(P)$, atunci când starea este x. Starea x' notează starea în care tranziționează automatul aflat în starea x, ca urmare a producerii evenimentului $e(P)$. Cu alte cuvinte, problema tranziției dintr-o stare x în starea x' se pune numai pentru evenimente posibile în starea x.

Oricărui automat i se poate ataşa o ieşire, situație când automatul este definit printr-un septuplu (E, X, P, f, x_0, Y, g), unde E, X, P, f şi x_0 îşi păstrează semnificațiile definite anterior, la care se adaugă obiectele definite mai jos:

- Y este mulțimea ieşirilor,
- g este funcția ieşirii: $g : X \times E \to Y, y = g(x,e)$, definită numai pentru $e(P)$, atunci când starea este x, $g(x,e)$ nu este definită pentru $e(P)$.

Formulărilor matematice prezentate anterior le corespunde o reprezentare grafică de tip graf orientat, denumită diagrama de tranziții, în care nodurile au semnificația stărilor, iar fiecare arc orientat este etichetat cu evenimentul e care declanşează tranziția dintre cele două stări (x şi x') unite prin arcul respectiv.

În cazul când este definită şi ieşirea automatului, arcul orientat ce porneşte din nodul x şi corespunde evenimentului $e(P)$ va primi o a doua etichetă cu valoarea ieşirii $y=g(x,e)$, în plus față de eticheta e. Facem precizarea că în aceste reprezentări grafice, restricția aplicațiilor f şi g la evenimentele posibile în starea $x=e(P)$ se traduce prin faptul că din nodul x al grafului, nu pornesc arce corespunzând tuturor elementelor lui $E(P)$, ci numai arce corespunzând elementelor din mulțimea E dependentă de x.

4.5.2 Paralelă între automate şi reţele Petri

Pentru un studiu mai aprofundat al modelării sistemelor dinamice cu evenimente discrete prin automate şi reţele Petri pot fi consultate referinţele bibliografice [62], [81], [87], [88] şi [106].

Concluzia principală a comparaţiei celor două modele este că, sub aspect teoretic, modelul de tip reţea Petri este echivalent cu modelul de tip automat. Aceasta înseamnă că, pornind de la un sistem dinamic cu evenimente discrete, care are o semnificaţie fizică, vom putea construi ambele tipuri de modele. Din punct de vedere practic, există însă unele aspecte care pot justifica faptul că unul din aceste două tipuri de modele este mai potrivit şi mai profitabil pentru studierea comportării sistemului. Aceste aspecte vor fi prezentate în acest paragraf.

În general, alegerea unui anumit tip de model rămâne la latitudinea expertului care realizează modelarea, depinzând de experienţa şi de abilităţile acestuia [49]. Există totuşi unele criterii generale, orientative, care trebuie înţelese în contextul unor situaţii reale frecvent întâlnite.

O primă concluzie este că modelele de tip automat sunt recomandate atunci când luăm în considerare evenimente care se petrec în exteriorul sistemului fizic modelat iar dacă ne interesează evidenţierea unor evenimente interne sistemului fizic, este preferabil să se facă apel la reţelele Petri. În cazul în care modelul trebuie să permită operarea cu un număr mare de stări distincte, reţelele Petri pot fi mai avantajoase, întrucât diferenţierea între stări se realizează prin marcajul reţele şi nu prin extinderea topologiei. Extinderea topologiei este o situaţie inevitabilă în cazul diagramelor de tranziţii asociate automatelor cu un număr mare de stări [86]. Altfel spus, efortul de modelare în cazul discutat este mai mic dacă se caută o topologie de reţea Petri relativ simplă cu mecanismul aferent pentru asignarea şi modificarea marcajului, decât dacă se generează o diagramă de tranziţii cu topologie relativ complexă şi complicată.

Un ultim aspect se referă la posibilitatea asamblării unui model din sub-modele. În principiu, utilizarea reţelelor Petri permite această facilitate de asamblare din sub-modele, oferind o mai mare flexibilitate în captarea unor detalii ale modelului. Utilizarea automatelor este mai rigidă, întrucât modelul trebuie privit de la bun început în ansamblul său şi se obţine mai greu modularizarea în sub-modele.

Pentru modelarea cronicilor de alarme în sensul determinării situaţiilor de consecutivitate şi concurenţă am ales reţelele Petri. Pentru scopul propus, de a analiza posibilele relaţii între cronicile frecvente de alarme, este suficient să folosim reţele Petri reduse, având maxim 1 jeton în poziţiile reţelei. În acest sens, nu vom folosi întreg potenţialul de modelare oferit de reţelele Petri, deoarece este suficient să folosim proprietăţile reţelelor Petri reduse.

Capitolul 5. EXTRAGEREA DE REGULI DE ASOCIERE

În ultimii ani, regulile de asociere sunt studiate ca o alternativă la sistemele expert, deoarece extragerea de reguli de asociere permite o achiziţie de informaţii mai uşor de realizat decât în cazul sistemelor expert [11],[52], [64], [80].

Extragerea de reguli de asociere este un domeniu important pentru prelucrarea informaţiilor în scopul descoperirii de cunoştinţe. În cadrul unui sistem de evenimente discrete, obiectivul principal al extragerii de reguli de asociere este de a descoperi cunoştinţe care sunt potenţial utilizabile în procesele de supervizare a acestor sisteme. Cunoştinţele descoperite pot fi secvenţe de evenimente, tipare sau cronici. În literatura de specialitate, în anumit cazuri se foloseşte termenul de *recunoaştere* sau *descoperire* în loc de termenul *extragere*.

Aşa cum este indicat în lucrările [8] şi [9] procesul de recunoaştere de reguli de asociere este constituit din următoarele etape:

- Specificarea unei descrieri corecte a domeniului de cunoştinţe, prin definirea cunoaşterii iniţiale de informaţii relevante şi definirea scopurilor utilizatorului,
- Generarea unei mulţimi de informaţii de tip obiective sau focalizarea atenţiei asupra unei mulţimi de informaţii asupra cărora se va efectua recunoaşterea,
- Preprocesarea informaţiilor pentru a reduce dimensiunea mulţimii de informaţii analizate şi transformarea lor într-o formă mai simplă pentru procesul de analiză a informaţiilor,
- Alegerea şi aplicarea algoritmului de recunoaştere,
- Evaluarea şi consolidarea soluţiilor finale obţinute.

Principalele direcţii de cercetare sunt organizate funcţie de tipul de cunoştinţe care sunt analizate în vederea recunoaşterii de reguli de asociere.

În funcţie de tipul de informaţii la care se referă, recunoaşterea de reguli de asociere poate fi clasificată astfel:

- Recunoaşterea de *episoade frecvente* de evenimente, în scopul determinării unor mulţimi parţial ordonate de evenimente de diverse tipuri [80] ,
- Recunoaşterea unor *generalizări de date*, în scopul determinării unor colecţii de caracteristici esenţiale ale informaţiilor analizate [11]. Recunoaşterea generalizărilor de date foloseşte mai multe tehnici de procesare: abstractizare, procesare analitică, caracterizare etc.
- Recunoaşterea de *reguli de clasificare*, are ca rezultat clasificarea seturilor de date pornind de la valorile anumitor atribute,

- Recunoaşterea unor *colecţii* are ca rezultat gruparea datelor în colecţii cu ajutorul unor măsuri de similitudine, după principiul maximizării şi minimizării. De exemplu, am dezvoltat în cadrul unui studiu o metodă de recunoaştere pentru identificarea unor forme geometrice prin analiza imaginilor bi-dimensionale folosind codul Freeman şi logica fuzzy [112],
- Recunoaşterea unor *cunoştinţe de evoluţie* constă în definirea datelor cu caracter temporal într-o proporţie importantă în cadrul datelor stocate în baza de date. Căutarea de regularităţi poate facilita predicţia apariţiei de evenimente cu factor de risc ridicat, descoperirea de legături cauzale şi tendinţe în evoluţia unor procese sau activităţi. Abordarea cea mai frecventă este cea bazată pe cronici [45], [46],
- Recunoaşterea unor *tipare de acces*; aceste cunoştinţe sunt specifice mediilor distribuite ce furnizează informaţii în care entităţile sunt legate între ele pentru a facilita accesul interactiv; capturarea unor astfel de tipare de acces ale utilizatorilor poate conduce la concluzii despre optimizarea organizării informaţiei furnizate.

5.1 Recunoaşterea regulilor de asociere

Considerând a_i şi b_j perechi de informaţii de tip atribut şi valoare, asociate între ele cu o frecvenţă mare în mulţimea de informaţii analizată, definim regulile de asociere ca fiind implicaţii logice de tipul :

$$a_1 \cap ... \cap a_i \Rightarrow b_1 \cap ... \cap b_j \qquad (5.1)$$

Aceste implicaţii logice sunt caracterizate de doi parametri principali: gradul lor de încredere şi suportul lor în baza de date a acestor informaţii atribut-valoare.

Regulile de asociere se definesc pe un număr de elemente $I = \{i_1, i_2, ..., i_n\}$. Fie D o mulţime de tranziţii, unde fiecare tranziţie T se constituie ca o submulţime a lui I, prin relaţia $T \subseteq I$. O observaţie importantă este aceea că vom lua în considerare prezenţa elementelor în tranziţie şi nu vom considera alte caracteristici cantitative sau calitative ale elementelor. Fiecare tranziţie are asociat un identificator.

Suportul unei submulţimi X al elementelor I din mulţimea de tranziţii D, notat cu $\Delta(X)$, se calculează ca fiind procentul de tranziţii T din D pentru care $X \subseteq T$:

$$\Delta(X) = \frac{\left|\{T \in D | X \subseteq T\}\right|}{|D|} \cdot 100[\%] \qquad (5.2)$$

Submulțimea X a lui I este frecventă dacă suportul ei depășește o valoare minimă :

$$\Delta(X) \geq f_{\min} \qquad (5.3)$$

O regulă de asociere $\Re : X \rightarrow Y$ este o implicație pentru care $X \subseteq I$, $Y \subseteq I$ și $X \cap Y \neq \emptyset$.

Suportul unei reguli de asociere $\Re$, notat $\Delta(\Re)$ se calculează ca fiind numărul de tranziții T din D pentru care $X \cup Y \subseteq T$. În raport cu X și Y, suportul unei reguli de asociere se notează:

$$\Delta(\Re) = \Delta(X \cup Y) \qquad (5.4)$$

Coeficientul de încredere al unei reguli de asociere $\Re$, notat $conf(\Re)$ se calculează ca procentul de tranziții T din D pentru care dacă $X \subseteq T$ și $Y \subseteq T$, $X \cup Y \subseteq T$, probabilitatea condițională ca o tranzacție să îl conțină pe Y dacă îl conține pe X :

$$conf(\Re) = \frac{\Delta(X \cup Y)}{\Delta(X)} \cdot 100[\%] \qquad (5.5)$$

Noțiunea de suport indică frecvența de apariție (numărul de tranziții) a unui anumit tipar în mulțimea de tranziții D. Coeficientul de încredere (procent) indică forța relației dintre elementele regulii de asociere [67].

Cel mai adesea sunt căutate regulile cu suport și grad de încredere mari, acestea fiind denumite și *reguli puternice*. Astfel, procesul de descoperire constă din doi pași:

- primul pas îl constituie descoperirea seturilor de elemente frecvente care au un suport suficient de mare,
- al doilea pas îl constituie folosirea lor pentru derivarea de reguli puternice.

Odată parcurs primul pas, care este foarte mare consumator de resurse informatice, cel de-al doilea pas se rezolvă fără o dificultate de calcul deosebită.

Folosirea seturilor frecvente pentru derivarea de reguli puternice se efectuează în baza unei proceduri de calcul care determină parametrii regulii $\Re : X \rightarrow Y$ pentru fiecare set frecvent X și pentru fiecare submulțime $Y \subset T$.

5.2 Proprietăţile seturilor de elemente

În acest paragraf sunt introduse teoremele care descriu proprietăţile seturilor de elemente, aşa cum sunt ele prezentate în lucrările [11], [41] şi [82]:

Teorema 5.1

Suportul submulţimilor depinde de relaţiile dintre submulţimi:

Dacă o submulţime Y este inclusă într-o submulţime X, ceea ce se exprimă prin relaţia $Y \subseteq X$, atunci suportul submulţimilor respectă relaţia $\Delta(Y) \geq \Delta(X)$, deoarece toate tranziţiile din D ce îl conţin pe X îl conţin în mod necesar şi pe Y.

Teorema 5.2

Mulţimile de elemente care nu sunt frecvente determină supramulţimi care nu sunt frecvente:

Dacă un set de elemente X nu are suportul minim necesar pe D, adică $\Delta(X) \leq f_{\min}$, atunci orice supramulţime $Y \supseteq X$ nu va avea suportul minim necesar, deoarece $\Delta(Y) \leq \Delta(X) \leq f_{\min}$.

Teorema 5.3

Seturile de elemente frecvente determină submulţimi frecvente:

Dacă setul de elemente Y este frecvent pe D, deci $\Delta(Y) \geq f_{\min}$, atunci orice submulţime $X \subseteq Y$ va fi frecventă deoarece $\Delta(Y) \geq \Delta(X) \geq f_{\min}$.

5.3 Proprietăţile regulilor de asociere

Aşa cum este indicat în lucrările [11], [41] şi [82], teoremele referitoare la proprietăţile regulilor de asociere sunt următoarele:

Teorema 5.4

Regulile de asociere nu se compun în reguli antecedente:

Dacă $X \rightarrow Y$ şi $Y \rightarrow Z$ nu este neapărat adevărat că $X \cup Y \rightarrow Z$, deoarece reuniunea la nivelul elementelor înseamnă intersecţie la nivelul mulţimilor de tranziţii ce asigură suportul pentru X şi Y, intersecţia acestora putând fi mulţimea vidă $\varnothing$.

Teorema 5.5

Regulile de asociere nu se descompun în reguli antecedente:

Dacă $X \cup Y \rightarrow Z$ nu este neapărat adevărat că $X \rightarrow Y$ şi $Y \rightarrow Z$, deoarece în acest caz restricţiile de suport rămân valabile, dar nu avem controlul restricţiilor de grad

de încredere, dat fiind faptul ca suportul pentru X sau pentru Y poate fi considerabil mai mare decât suportul reuniunii lor, ceea ce micşorează gradul de încredere al noilor reguli.

Teorema 5.6

Regulile de asociere se pot descompune în reguli consecvente:

Dacă $X \to Y \cup Z$ atunci $X \to Y$ şi $Y \to Z$, deoarece dacă $X \cup Y \cup Z$ este set frecvent orice submulţime a lui este frecventă, iar la nivelul gradului de încredere raportul creşte, deci regulile au parametri mai buni; totuşi acest aspect nu este foarte interesant în contextul recunoaşterii de reguli pentru că se doreşte obţinerea de reguli din ce în ce mai puternice din reguli mai slabe şi nu invers.

Teorema 5.7

Regulile de asociere nu sunt tranzitive:

Dacă $X \to Y$ şi $Y \to Z$ nu se poate spune că $X \to Z$, deoarece, în primul rând mulţimile ce asigură suportul celor două reguli, adică $X \cup Y$ şi $Y \cup Z$, nu conduc la nici o concluzie în privinţa tranzitivităţii.

Teorema 5.8

Regulile de asociere sunt inferenţe:

Dacă regula $(Z \cup X) \to X$ este valabilă atunci $\forall Y \subseteq X, Y \neq \emptyset$ obţinem inferenţa $(Z \cup Y) \to Y$. Suportul ambelor reguli este acelaşi $\Delta(Z)$, gradul de încredere creşte deoarece $(Z \cup Y) \supseteq (Z \cup X)$, deci suportul aflat la numitor scade. De asemenea dacă regula $X \to (Z \cup X)$ nu este valabilă atunci $\forall Y \subseteq X, Y \neq \emptyset$ nu este valabilă nici regula $Y \to (Z \cup Y)$ deoarece suportul ambelor reguli este acelaşi $\Delta(Z)$ şi gradul de încredere scade deoarece $Y \subseteq X$.

5.4 Analiza formală a conceptelor

În acest paragraf, vom prezenta corelaţia dintre extragerea de asociaţii şi analiza formală a conceptelor pe baza lucrării [59]. Pentru aceasta vom introduce câteva elemente din teoria laticilor.

Definiţia 5.1

Tripletul $(L, \wedge, \vee)$, unde L este o mulţime nevidă, iar $\wedge$ şi $\vee$ sunt două operaţii binare pe L se numeşte ***latice****, dacă pentru $\forall a, b, c \in L$ sunt satisfăcute următoarele axiome:*

- comutativitate $a \wedge b = b \wedge a$ şi $a \vee b = b \vee a$ ***(5.6)***

- asociativitate $a \wedge (b \wedge c) = (a \wedge b) \wedge c$ şi $a \vee (b \vee c) = (a \vee b) \vee c$ **(5.7)**
- absorbţie $a \wedge (a \vee b) = a$ şi $a \vee (a \wedge b) = a$ **(5.8)**

Definiţia 5.2

O latice $(L, \wedge, \vee)$ se numeşte latice completă dacă $\forall X, X \subseteq L$ există:

$$\bigcup_{x \in X} x \text{ şi } \bigcap_{x \in X} x \qquad \textbf{(5.9)}$$

Definiţia 5.3

Fie $(L_1, \wedge_1, \vee_1)$ şi $(L_2, \wedge_2, \vee_2)$ două latici. O aplicaţie $f: L_1 \to L_2$ se numeşte homomorfism de latice dacă păstrează operaţiile, adică:

$f(a \wedge_1 b) = f(a) \wedge_2 f(b), \ \forall a, b \in L_1;$ **(5.10)**

$f(a \vee_1 b) = f(a) \vee_2 f(b), \ \forall a, b \in L_1.$ **(5.11)**

Definiţia 5.4

Fie $(L_1, \wedge_1, \vee_1)$ şi $(L_2, \wedge_2, \vee_2)$ două latici. O aplicaţie $f: L_1 \to L_2$ se numeşte izomorfism dacă există un homomorfism $g: L_2 \to L_1$ astfel încât:

$f \circ g = 1_{L_2}$ şi $g \circ f = 1_{L_1}$ **(5.12)**

Definiţia 5.5

Fie I o mulţime de informaţii şi $c: N(S) \to N(S)$ o aplicaţie pe mulţimea părţilor lui S. Operatorul c se numeşte operator de închidere pe S dacă pentru $\forall X, Y \subseteq S$ satisface următoarele proprietăţi:

- extensie $X \subseteq c(X)$ **(5.13)**
- monotonie *dacă* $X \subseteq Y$ *atunci* $c(X) \subseteq c(Y_0)$ **(5.14)**
- idempotenţă $c(c(X)) = c(X)$ **(5.15)**

Definiţia 5.6

O submulţime X a lui S se numeşte mulţime închisă dacă operatorul său de închidere conţine toate elementele submulţimii S.

$c(X) = X$ **(5.16)**

Definiţia 5.7

Un context este un triplet (G, M, I), unde G şi M sunt mulţimi şi $I \subseteq G \times M$. Elementele lui G sunt numite obiecte iar cele ale lui M sunt numite atribute. Pentru orice $g \in G$ şi $m \in M$ se notează *gIm* relaţia dintre *g* şi *m* adică $(g, m) \in I$.

Fie *(G,M,I)* un context. Atunci mapările *s* şi *t* prezentate mai jos definesc o conexiune Galois între mulţimile părţilor lui *G* respectiv *M*:

$s: \wp(G) \rightarrow \wp(M)$, $s(X)=\{m \in M \mid \forall g \in X, gIm\}$ *(5.17)*

$t: \wp(M) \rightarrow \wp(G)$, $t(Y)=\{g \in G \mid \forall m \in Y, gIm\}$ *(5.18)*

Mulţimea $s(X)$ reprezintă setul tuturor atributelor comune obiectelor din X iar mulţimea $t(Y)$ reprezintă setul tuturor obiectelor comune atributelor din Y.

Definiţia 5.8

Un concept al contextului (G,M,I) este definit ca perechea (X,Y) unde $X \subseteq G$ şi $Y \subseteq M$ pentru care $s(X)=Y$ şi $t(Y)=X$.

Altfel spus, un concept este o pereche de mulţimi închise (X,Y) deoarece $X=t(Y)=t(s(X))$. X se numeşte *extensiune* a conceptului iar Y se numeşte *intensiune* a conceptului.

Definiţia 5.9

Un concept generat de un singur atribut $m \in M$, care se determină prin relaţia $\alpha(m)=(t(\{m\}),c(\{m\}))$ se numeşte *concept-atribut*.

Definiţia 5.10

Un concept generat de un obiect $g \in G$, care se determină prin relaţia $\beta(g)=(c(\{g\}),s(\{g\}))$, se numeşte *concept-obiect*.

Mulţimea tuturor conceptelor unui context se notează cu $\beta(G,M,I)$.

Teorema 5.9

Un concept (X_1,Y_1) este un subconcept al lui (X_2,Y_2), notat prin $(X_1,Y_1) \leq (X_2,Y_2)$ dacă şi numai dacă $X_1 \subseteq X_2$ şi $Y_2 \subseteq Y_1$.

Definiţia 5.11

O submulţime P al unei mulţimi ordonate Q se numeşte densă faţă de un operator binar <op>, dacă $\forall q \in Q$, există $Z \subseteq P$, astfel încât $q =$ <op> Z.

Enunţăm teoremele fundamentale ale analizei formale a conceptelor [77] :

Teorema 5.10

Fie (G,M,I) un context; atunci $\beta(G,M,I)$ este o latice completă cu operatorii $\wedge$, $\vee$ calculaţi astfel:

$$\bigvee_j (X_j,Y_j) = (c(\bigcup_j X_j), \bigcap_j Y_j), \quad (5.19)$$

respectiv:

$$\Lambda_j(X_j, Y_j) = (\bigcap_j X_j, c(\bigcup_j Y_j)). \qquad (5.20)$$

Teorema 5.11

Reciproc, dacă L este o latice completă, L este izomorfică faţă de $\beta(G,M,I)$ dacă şi numai dacă există mapările $\gamma: G \to L$ şi $\mu: M \to L$, astfel încât $\gamma(G)$ este $\vee$-dens în L, $\mu(M)$ este $\wedge$-dens în L, şi gIm este echivalent cu $\gamma(g) \leq \mu(m)$ pentru orice $g \in G$ şi $m \in M$ şi L este izomorfic cu $\beta(L,L,\leq)$. Laticea $\beta(G,M,I)$ este denumită laticea contextului.

5.5 Exemplu de extragere de reguli de asociere

Considerăm o listă de evenimente sau de alarme $\{a, b, c, d, e\}$ şi o bază de tranziţii T care conţine toate cronicile frecvente de alarme recunoscute prin aplicarea algoritmului descris în Capitolul 2. Baza de tranziţii T considerată pentru exemplificarea extragerii de reguli de asociere este prezentată în *Tabelul 5.1* :

Tranziţii	*Cronici frecvente de alarme*
1	*a b d e*
2	*b c e*
3	*a b d e*
4	*a b c e*
5	*a b c d e*
6	*b c d*

Tabelul 5.1 Baza de tranziţii *T*

Din analiza bazei de tranziţii prezentate în tabelul de mai sus, putem extrage următoarele reguli de asociere :

- Regulile de asociere având confidenţa $conf(\Re) = 100\%$, calculate conform relaţiei **(5.5)**:

$(a \to b)$	$(a \to be)$	$(ab \to e)$	$(ad \to be)$	$(abd \to e)$
$(a \to e)$		$(ad \to b)$	$(de \to ab)$	$(ade \to b)$
$(c \to b)$		$(ad \to e)$		$(bde \to a)$
$(d \to b)$		$(ae \to b)$		
$(e \to b)$		$(ce \to b)$		
		$(de \to a)$		

$(de \rightarrow b)$	**(5.21)**

- Regulile de asociere având confidența $conf(\Re) \geq 80\%$, calculate conform relației **(5.5)**:

$(e \rightarrow a)$ $(e \rightarrow ab)$	
$(b \rightarrow e)$	**(5.22)**

Interpretarea logică a acestor reguli de asociere poate fi făcută astfel:

$(ab \rightarrow e)$ cu confidența 100% : atunci când apar alarmele *a* şi *b* apare întotdeauna şi alarma *e* (cu probabilitate 100%).

$(e \rightarrow ab)$ cu confidența ≥80% : alarma *e* apare cel puțin în proporție de 80% odată cu alarmele *a* şi *b*.

Seturile frecvente $\Delta(X) \geq f_{\min}$, cu frecvența minimă $f_{min}=50\%$ sunt descrise în *Tabelul 5.2* :

Cronici de alarme	*Frecvență*
{b}	100%;
{e}{b e}	84%;
{a} {c} {d} {a b} *{a e} {b c} {b d}* *{a b e}*	68%;
{a d} {c e} {d e} *{a b d} {a d e}* *{b c e} {b d e}* *{a b d e}*	50%.

Tabelul 5.2 Seturi frecvente

În *Figura 5.1* este reprezentată diagrama Hasse [132] a laticei contextului pentru exemplul prezentat. Etichetele asociate conceptelor sunt obiectele, în cazul conceptelor-obiect, atributele, în cazul conceptelor-atribut, în celelalte cazuri reprezentând diferențele specifice dintre concepte.

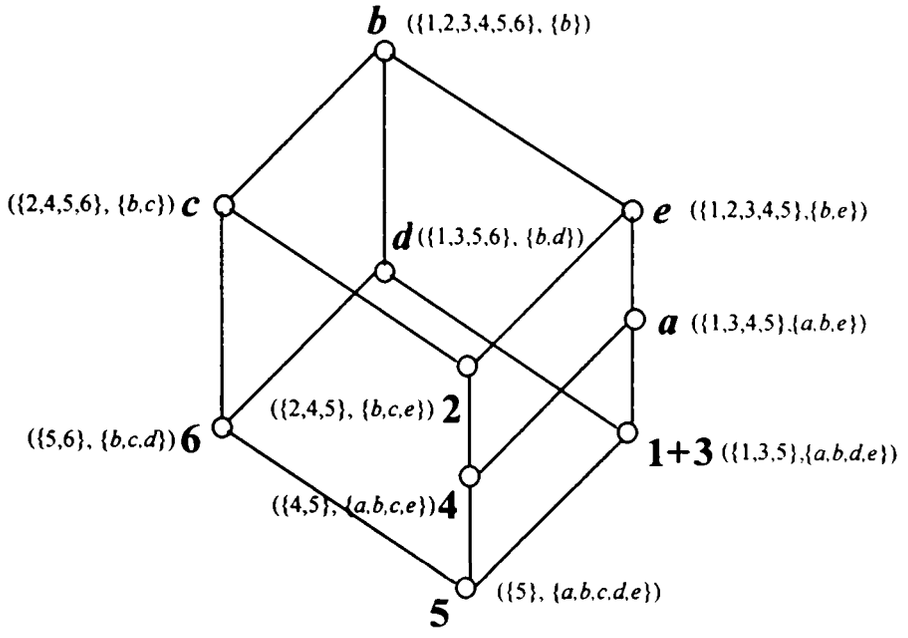


Figura 5.1 Diagrama Hasse a unei latici

Conceptele frecvente sunt perechile *(X, Y)* unde $|X| \geq |D| * f_{\min}$.

Figura 5.2 reprezintă o reducere a diagramei lui Hasse la conceptele frecvente:

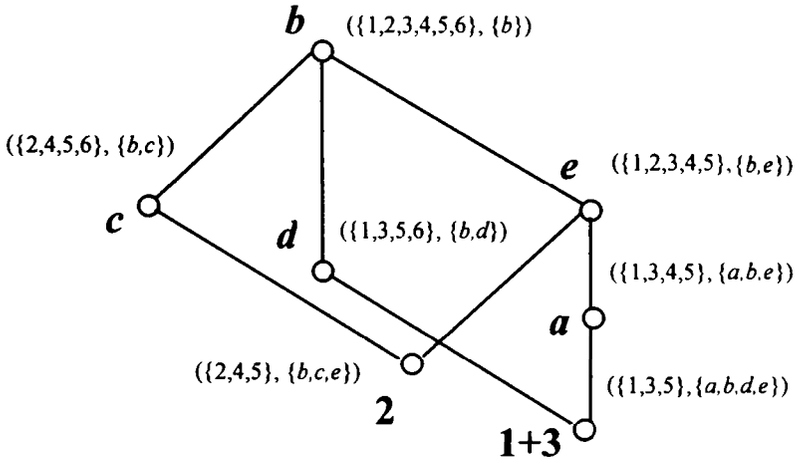


Figura 5.2 Diagrama Hasse redusă la dimensiunea cronicilor frecvente

Din analiza diagramei Hasse a seturilor frecvente se determină că seturile frecvente maximale sunt *{b c e}* şi *{a b d e}*.

5.6 Generarea de reguli

Regulile de asociere au fost propuse prima dată în [9]. Totuşi din cele ce vor urma se poate observa că ele reprezintă acele implicaţii parţiale, cu constrângerile de suport şi grad de încredere, propuse în [59].

Fie *(G,M,I)* un context; o regulă de implicaţie parţială este un triplet *(X,Y,p)* unde $X,Y \subseteq M$ sunt mulţimi de atribute, iar $p=P(Y|X)$.

Regulile de asociere sunt deci reguli de implicaţie parţială cu constrângerile de suport $S(X \cup Y) \geq min\_sup$ şi $p \geq min\_conf$.

Fie I o mulţime de implicaţii parţiale; o regulă (K,L,p) poate fi derivată din I, notat $I: K \xrightarrow{p} L$, dacă regula poate fi obţinută din I aplicând anumite reguli de inferenţă; în acest caz regula se numeşte *redundantă* în raport cu I.

O mulţime $G \subseteq I$ se numeşte *generatoare* pentru I dacă şi numai dacă $G \cap I$. O mulţime generatoare minimală B se numeşte bază. Cum mulţimea de reguli de asociere poate fi de dimensiuni mari, poate fi interesant să putem găsi o bază pentru ea. Aceasta înseamnă că numai un set de reguli restrâns şi mai uşor inteligibil va fi prezentat utilizatorului, ce apoi va putea deriva şi alte reguli interesante. Setul de implicaţii poate fi separat în implicaţii globale, având $p=1$ şi implicaţii strict parţiale, având $p<1$.

Mulţimea $X \subseteq M$ se numeşte *pseudo-intensiune* dacă $X \neq c(X)$ şi pentru $\forall Q \subset X$, Q pseudo-intensiune, $c(Q) \subset X$. Mulţimea $\{X \rightarrow c(X) \setminus X \mid X$ pseudo-intensiune$\}$ este o bază pentru toate implicaţiile globale.

In exemplul *{a, b, c, d, e}* este mulţimea de pseudo-intensiuni. O bază pentru implicaţiile globale va fi $b=\{a \rightarrow \{b,e\}, c \rightarrow b, d \rightarrow e, e \rightarrow b, \{b,d,e\} \rightarrow a\}$. Orice altă implicaţie globală poate fi derivată din acestea folosind câteva reguli simple de inferenţă. Pentru găsirea unei baze a implicaţiilor strict parţiale, trebuie observat în primul rând că pentru $\forall Z \subseteq X$, $X \xrightarrow{p} Y$ dacă şi numai dacă $X \xrightarrow{p} Y \cup Z$. De aceea vom discuta doar regulile pentru care $X \subseteq Y$, pentru care de altfel şi $c(X) \xrightarrow{p} c(Y)$, deci ne vom referi la reguli pentru care atât X cât şi Y sunt intensiuni ale unui concept.

Fie M_1, M_2 şi M_3 intensiuni cu $M_1 \subseteq M_2 \subseteq M_3$, atunci dacă $M_1 \xrightarrow{p} M_2$ şi $M_2 \xrightarrow{q} M_3$ atunci $M_1 \xrightarrow{pq} M_3$.

Să considerăm acum diagrama Hasse în care muchiilor le sunt asociate preciziile, ca în *Figura 5.2*. Muchia dintre conceptele etichetate b şi e corespunde implicaţiei $b \xrightarrow{5/6} e$. Implicaţia inversă are, în mod evident, precizia 1. Din diagramă trebuie considerate doar implicaţiile dintre concepte adiacente deoarece celelalte pot fi derivate.

Fie $I' \subseteq I(N(G,M,I))$ o submulţime a implicaţiilor parţiale stricte, putem defini graful orientat $G(I')=(V,E)$, unde vârfurile sunt reprezentate de intensiunile din M ce

participă la formarea implicaţiilor din I', iar muchiile sunt reprezentate de implicaţiile existente în I' între intensiuni.

Dacă există cicluri în graful $G(I')$ atunci există cel puţin o implicaţie parţială redundantă în I'.Ca o consecinţă a acestui fapt, câte o regulă din fiecare ciclu al grafului este redundantă şi poate fi eliminată. O caracterizare mai precisă a mulţimii generatoare a setului de implicaţii parţiale este următoarea:

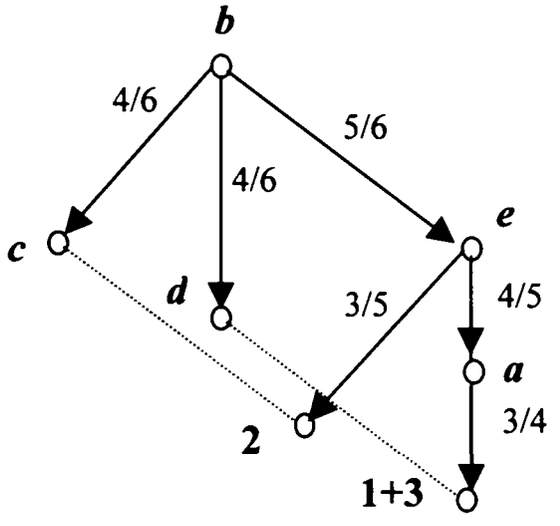


Figura 5.3 Diagrama Hasse în plan

I' este mulţime generatoare pentru $I(N(G,M,I))$ dacă $G(I')$ este un arbore de acoperire şi mulţimea M apare în consecventul unei singure implicaţii strict parţiale.

În *Figura 5.3* este prezentată o mulţime generatoare pentru tot setul de implicaţii strict parţiale al exemplului dat. Precizia unei reguli redundante se obţine înmulţind preciziile regulilor implicate în ciclul pe care ea l-ar forma, preciziile trebuind inversate pentru implicaţiile de sens contrar (cele între un subconcept şi un concept).

De exemplu, precizia implicaţiei dintre C şi 2 se obţine înmulţind inversul preciziei implicaţiei dintre B şi C cu precizia implicaţiei dintre B şi E şi cea dintre E şi 2, adică $6/4*5/6*3/5=3/4$.

Pentru a obţine regulile cu restricţionate de *min_conf* se elimină muchiile cu precizie mai mică.

Un element $x \in L$ din laticea L se numeşte $\vee$-ireductibil ($\wedge$-ireductibil) dacă în diagrama Hasse are exact un vecin căruia i se *subsumează*.

Fie $V(E)$ şi $\Lambda(E)$ mulţimile de elemente ireductibile faţă de cele două operaţii; orice latice finită este unic determinată (inclusiv la *izomorfism*) de aceste două mulţimi. În cazul nostru, $V(E)=\{2; 4; 6; 1,3\}$ şi $\Lambda(E)=\{a, c, d, e\}$. Totuşi nu trebuie ignorat faptul că un concept în contextul de faţă este o pereche intensiune-extensiune, şi deci încercarea de a păstra doar elementele acestor două mulţimi va eşua deoarece conceptele trebuie

memorate în totalitatea lor, altfel conceptele eliminate nu vor mai putea fi recuperate prin operațiile specifice laticii. Următoarea teoremă oferă limite superioară și inferioară pentru o bază a mulțimii implicațiilor strict parțiale :

Dacă submulțimea $I' \subseteq I(K=\beta(G,M,I))$ este o bază, atunci:

$$\frac{|\vee(K) \cap \wedge(K)|}{2} \leq |I'| \leq |K-1| \qquad (5.23)$$

Limita superioară este foarte strânsă pentru cea mai mare parte a laticilor deci construirea unei baze nu va diminua spațiul de memorare față de cel necesar pentru mulțimea generatoare. Nici limita inferioară nu este utilizată deoarece există latici pentru care aceasta este *0*. În concluzie mulțimea generatoare ar trebui să fie suficientă pentru a înlocui o bază.

Combinând baza determinată anterior pentru mulțimea implicațiilor globale și mulțimea generatoare pentru implicațiile parțiale, cu constrângerile respective, se obține următoarea mulțime generatoare pentru regulile de asociere din exemplul de tranziții prezentat în *Tabelul 5.1*:

$c \xrightarrow{1} b$ $\quad a \xrightarrow{1} \{b,e\}$ $\quad \{b,d,e\} \xrightarrow{1} a$

$d \xrightarrow{1} b$

$e \xrightarrow{1} b$

$b \xrightarrow{5/6} e$

$e \xrightarrow{4/5} a$ $\qquad (5.24)$

Capitolul 6. PROIECTUL INFORMATIC

6.1 Specificații de proiectare

Pornind de la algoritmul de recunoaştere a cronicilor frecvente din jurnalele de alarme de telecomunicații, prezentat anterior în *Capitolul 2*, am dorit să realizez un proiect informatic care să aplice acest algoritm. Astfel, am dezvoltat un proiect informatic într-un mediu de programare deschis care integrează procedurile informatice de generare a cronicilor candidate, calculează frecvențele de apariție ale cronicilor candidate şi construieşte soluția problemei prin asamblarea cronicilor frecvente.

După cum am arătat în *Capitolul 4*, pornind de la noțiunile teoretice ale rețelelor Petri, am dezvoltat un modul informatic pentru construirea unei rețele Petri la care tranzițiile sunt etichetate de cronicile frecvente descoperite de algoritmul realizat anterior. Aceste rețele Petri pot fi analizate în raport cu proprietățile lor şi cu extragerea de reguli de asociere, rezultând astfel o soluție completă pentru analiza alarmelor.

În final, am realizat mai multe module informatice pentru vizualizarea rezultatelor către un sistem de supervizare, prin reprezentarea grafică a rețelei Petri construite în urma analizei cronicilor frecvente recunoscute şi prezentarea acestora sub o formă interpretabilă de către un sistem expert de supervizare. Menționez faptul că rolul proiectului informatic care l-am dezvoltat nu a fost de a se substitui unui sistem expert de supervizare a unei rețele de telecomunicații ci de a analiza fluxul de alarme care este prezentat unui astfel de sistem informatic, prin asamblarea secvențelor de alarme care se repetă cu o anumită frecvență.

Pentru realizarea proiectului informatic am folosit regulile generale de management de proiect informatic. Am planificat fazele de analiză, programare şi testare pentru fiecare modul informatic dezvoltat. Am avut nevoie de un mediu de programare integrat care să fie flexibil şi să permită dezvoltarea ulterioară de module de procesare, pe măsură ce avansam cu analiza teoretică a problemei de procesare a alarmelor. Am ales mediul de dezvoltare *OMNeT++ (Objective Modular Network Testbed in C++)* deoarece acesta prezintă multe facilități de interfațare şi permite dezvoltarea de noi aplicații, în limbajul de programare *C++*, care să completeze posibilitățile de analiză şi sinteză a rezultatelor.

Aplicația informatică a fost dezvoltată în jurul algoritmului de recunoaştere a cronicilor frecvente din jurnalele de alarme de telecomunicații. În acest sens a trebuit să respect condițiile de colectare a alarmelor în timp real, ajungând uneori la anumite limitări ale mediului de programare utilizat. De asemenea, prin executarea unor teste de

performanţă, am eliminat majoritatea erorilor de programare care sunt conţinute, în general, în astfel de proiecte informatice atunci când creşte complexitatea interfeţelor de comunicare între modulele componente.

Pentru acest proiect informatic am programat o serie de noi module le-am integrat într-o bibliotecă de funcţii şi le-am făcut publice în regim *open-source* [137].

6.2 Mediul de programare OMNeT++

Mediul de programare pe care l-am ales pentru dezvoltarea proiectului informatic de analiză a jurnalelor de alarme este *OMNeT++ (Objective Modular Network Testbed in C++)*, conceput şi dezvoltat între anii 1996-2005 în colaborare între Departamentele de Telecomunicaţii de la Universitatea Tehnică din Budapesta, TU Delft şi Universitatea din Karlsruhe. OMNeT++ a fost dezvoltat iniţial de către Andràs Varga de la Universitatea Tehnică din Budapesta. Pentru prezentarea mediului de programare OMNeT++ se pot consulta articolele lui Varga [127] şi [128] şi manualul de utilizare [126].

OMNeT++ este un mediu de simulare cu utilizare generală pentru sistemele cu evenimente discrete, fiind dezvoltat pornind de la o tehnologie orientată pe obiecte. OMNeT++ poate fi folosit pentru mai multe tipuri de simulări printre care menţionez o serie de proiecte realizate şi disponibile pe site-ul web al comunităţii programatorilor OMNeT++[137]: sisteme cu coadă de aşteptare, protocoale de comunicaţii şi alte sisteme dinamice cu evenimente discrete (INET, Mobility Framework, IPv6Suite, MPLS, etc.)

În domeniul simulării de reţele există mai multe alte medii de programare, unele comerciale cum ar fi COMNET, OPNET, SES Workbench, Arena, Simscript, iar altele academice cum ar fi NetSim++ sau Smurph. Am ales OMNeT++ pentru că este cu distribuţie liberă şi foarte bine documentat pe site-ul comunităţii web [137]. Trebuie menţionat că, începând din Octombrie 2004, OMNeT++ are şi o versiune comercială, denumită OMNEST[138], destinată implementărilor comerciale ale unor aplicaţii informatice.

Avantajele principale oferite de OMNeT++ sunt următoarele :

- Nu este necesar să se studieze un nou limbaj specific de programare, programele pot fi realizate în C++,
- Se oferă o interfaţă grafică *GUI (Graphical User Interface)* foarte completă pentru urmărirea proceselor şi în procesul de verificare a funcţionalităţii programelor dezvoltate,
- Simulările sunt independente de platforma de dezvoltare şi sunt portabile pe diferite sisteme de operare *(unix, win32)*,
- Structurile pot fi modificate rapid fără a impacta codul sursă, folosind multiple posibilităţi de parametrare,

- Anumite clase predefinite şi multe alte biblioteci de funcţii sunt dezvoltate în continuare de către diverse persoane şi sunt publicate în regim liber, fapt care contribuie la creşterea continuă a facilităţilor de programare.

OMNeT++ oferă o arhitectură modulabilă, componentele acesteia fiind module dezvoltate în limbajul de programare C++ care pot fi asamblate în componente de nivel superior folosind un limbaj de nivel înalt *NED (Network Description Language).*

Limbajul NED implementat ca parte a mediului de programare OMNeT++ conţine mai multe facilităţi de programare şi de editare grafică pentru descrierea topologiei reţelei şi pentru parametrarea unor procese.

Componentele principale ale OMNeT++ sunt următoarele :

- Biblioteca centrală de simulare,
- Compilator pentru limbajul NED *(nedc)*,
- Interfaţă utilizator grafică pentru editarea topologiei reţelelor *(GNED)*,
- Interfaţă de simulare *(Tkenv)*,
- Interfaţă de tip linie de comandă pentru executarea simulărilor *(Cmdenv)*,
- Aplicaţie de vizualizare a graficelor rezultate în urma simulărilor *(Plove)*,
- Mai multe programe utilitare pentru realizarea de funcţii.

Modulele pot fi create dinamic în timpul unei simulări pentru a lua în considerare evoluţiile de topologie. Modulele pot avea un număr arbitrar de conexiuni dezvoltate pe principiul unor porturi de intrare-ieşire. Principiul porturilor permite elaborarea de modele complexe şi are avantajul de a defini module care pot fi reutilizate ulterior de alte aplicaţii informatice.

Principiul de comunicare este de a găsi portul de intrare care are prezent un mesaj şi de a trimite acel mesaj către portul de ieşire în urma validării unei condiţii de execuţie. În cazul nostru mesajele din porturi vor fi de fapt alarme sau cronici de alarme care sunt vehiculate sub forma unor jetoane între modulele de prelucrare a alarmelor.

Pentru verificarea funcţionalităţii programelor dezvoltate, mediul de simulare OMNeT++ prezintă toate facilităţile unui mediu de dezvoltare orientat pe obiecte : `watchdog`, `snapshot`, `breakpoint`, module de verificare a utilizării stivei, clase pentru colectarea rezultatelor, etc.

Limbajul de descriere a topologiei reţelei, numit *NED (Network Description Language)* conţine module simple care sunt declarate urmând apoi a fi implementate în clase C++, modulele având parametrii şi porturi:

```
#include "omnetpp.h"
class MyModule : public cSimpleModule
{
// macrocomandă pentru apelarea constructorului
// şi a relaţiilor de moştenire;
   Module_Class_Members (MyModule, cSimpleModule, 0)
```

```
    // funcţia utilizator poate fi definită în continuare:
    ...
};
    // anunţă această clasă ca fiind un modul integrat OMNeT:
Define_Module (MyModule);
```

Modulele definite de utilizator pot fi derivate prin tehnici standard de moştenire din programarea orientată pe obiecte. De exemplu :

```
#include "MyModule.h"
class MyDerivedModule : public MyModule
{
    Module_Class_Members (MyDerivedModule, MyModule, 0);
    // funcţia utilizator poate fi definită în continuare:
    ...
};
    // anunţă această clasă ca un modul derivat OMNeT:
Define_Module (MyDerivedModule);
```

Modulele de diferite tipuri pot fi asamblate împreună pentru a forma un modul nou, mai complex, numit modul compus, având următoarele proprietăţi :

- Din exteriorul unui modul compound modulele componente nu sunt vizibile,
- Modulele compuse se comportă la fel ca şi modulele simple,
- Modulele compuse nu implementează nici o funcţie suplimentară, ele oferind doar combinaţii ale facilităţilor modulelor simple componente,
- Permit structurarea ierarhică a programelor de simulare.

Deoarece OMNeT++ este în esenţă un mediu de dezvoltare adaptat simulării evenimentelor discrete, funcţionalitatea principală este dată de modulul de gestionare a evenimentelor. Modulul de gestionare a evenimentelor include tehnici de gestionare de tipul `nextevent time advance mechanism` şi foloseşte implicit biblioteca de funcţii de simulare. Evenimentele sunt generate de anumite module care trimit mesaje către alte module sau către ele însele (în acest fel se pot implementa temporizări de aşteptare de evenimente). Evenimentele sunt vehiculate între module sub forma unor mesaje.

Atunci când un mesaj ajunge la un modul se invocă metoda `handleMessage()`, care este o metodă virtuală oferită de `cSimpleModule`; `cSimpleModule` este o clasă care va trebui suprascrisă de către un modul utilizator pentru a implementa funcţionalităţi reale pentru gestionarea mesajelor. Metoda `handleMessage()` procesează mesajul şi trimite eventual alte mesaje către celelalte module care intervin în dialog. Pentru a putea accesa structura mesajelor care vor fi procesate, `handleMessage()` primeşte ca parametru un pointer către structura mesajului, reprezentat în general de un obiect `cMessage`. Prototipul metodei `handleMessage()` poate fi exprimat astfel:

```
void handleMessage (cMessage *)
```

Procesarea mesajelor depinde în general de starea unui anume modul (de exemplu dacă modulul este ocupat sau liber). Stările modulelor de procesare sunt membre ale claselor corespondente metodei `handleMessage()`. În mod uzual, membrele claselor sunt iniţializate în constructor. `cSimpleModule` oferă metoda virtuală `initialize()` ca un loc

preferențial pentru a parametra membrele claselor. Metoda `initialize()` este folosită pentru a genera anumite evenimente inițiale. `cSimpleModule` oferă de asemenea metoda `finish()` pentru a permite modulelor care s-au executat să înregistreze statistici despre simulare şi eventual să îşi curețe stivele sau tampoanele de colectare. La sfârşitul unei simulări, kernelul apelează o metodă `finish()` pentru toate modulele.

Prin metoda `handleMessage()`, orice modul dezvoltat în OMNeT++ poate decide să trimită un mesaj prin metode de tipul `send()`, să programeze trimiterea unui mesaj prin metode `scheduleAt()` sau să anuleze o programare prealabilă a unui mesaj prin metoda `cancelEvent()`. În mod uzual, metoda `cancelEvent()` este folosită pentru a anula o anumită temporizare de aşteptare care a fost în prealabil armată în aşteptarea unui mesaj.

Modulele OMNeT++ dispun de un număr arbitrat de porți pentru comunicare. Porțile pot fi identificate fie printr-un număr sau un index într-o listă de porți. Porțile pot fi numai unidirecționale, deci fie pentru recepția evenimentelor fie pentru transmisia evenimentelor între module.

Adăugarea de porți creşte complexitatea programului, dar permite utilizarea tehnicilor de reuzabilitate deoarece o poartă încapsulează cunoştințe de conectare cu alte module. OMNeT++ nu permite trimiterea de mesaje direct către alte module ci doar prin intermediul porților. Dacă s-ar trimite mesaje direct către module atunci în timpul simulării mesajele ar putea fi pierdute datorită dinamicii modulelor. Folosirea de porți de comunicație oferă servicii implicite de comunicații modulelor care le folosesc.

Structura de interconectare a modulelor prin conectori de porți de comunicație este reprezentată în *Figura 6.1*:

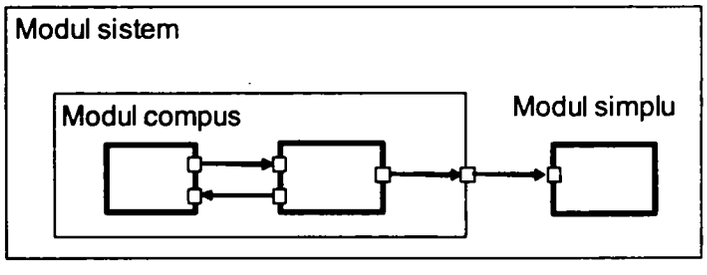


Figura 6.1 Structura modulară OMNeT++

OMNeT++ foloseşte un limbaj separat pentru specificarea topologiei unei rețele de module. Astfel, pentru fiecare program de simulare este necesară declararea unui fişier de configurare care descrie modulele simple şi compuse, conexiunile posibile între aceste module şi canalele de comunicație între porțile modulelor.

Canalele de comunicație reprezintă tipuri de conexiuni care pot fi parametrare prin declararea de întârzieri, rate de transmisie şi rate de eroare, ca de exemplu:

```
channel DialUpConnection
   delay normal (0.004, 0.0018)
   error 0.00001
```

```
    datarate 14400
endchannel
```

Modulele simple sunt declarate în NED prin parametri şi porţi de comunicaţie. Parametrii modulelor simple pot fi uşor accesaţi prin structuri C++ folosind metoda `par(nume.parametru)`. Porţile pot fi declarate prin specificarea tipului lor, de intrare sau de ieşire:

```
simple DataLinkProtocol
   parameters: …
   gates:
      in: from_upper_layer, from_physical_layer;
      out: to_upper_layer, to_physical_layer;
endsimple
```

Programarea structurată poate fi obţinută prin apelarea unor secvenţe de tipul unor condiţii `if` sau de tipul unor bucle `for` :

```
module Stochastic:
connections nocheck:
for i=0 .. 9 do
   Sender.outgate[i] _ Receiver[i].ingate if
   uniform(0,1) < 0.3;
endfor
endmodule
```

6.3 Arhitectura generală

Schema de principiu a aplicaţiei informatice pentru analiza jurnalelor de alarme este compusă din trei module specializate, prezentate în continuare :

- Modul de intrare are rolul de a citi jurnalele de alarme folosind interfaţa cu sistemul de colectare de alarme.
- Modulul de prelucrare a alarmelor în scopul extragerii de cronici de alarme frecvente
- Modul de asamblare a unei reţele Petri etichetată de cronicile de alarme determinate de către modulul anterior, având rolul de a finaliza analiza cronicilor frecvente.

Pentru exploatarea rezultatelor aplicaţiei informatice am definit câteva module de reprezentare grafică pentru vizualizarea benzilor de cronici frecvente şi pentru analiza performanţelor aplicaţiei (de exemplu pentru reprezentarea grafică a timpului de execuţie al algoritmului de recunoaştere a cronicilor frecvente pentru diferite frecvenţe minime şi diferite metode de generare a cronicilor candidate).

În *Figura 6.2* se prezintă arhitectura generală a aplicației informatice dezvoltate în mediul de programare OMNeT++ :

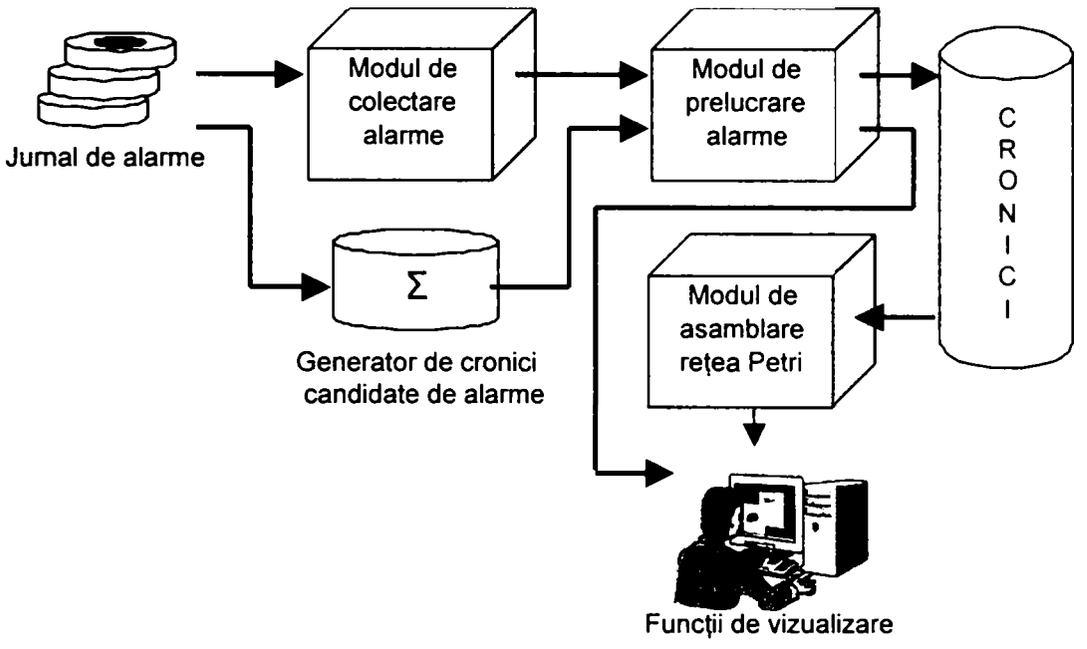


Figura 6.2 Schema de implementare a modulelor pentru recunoaşterea cronicilor

6.4 Modulul de colectare a alarmelor

Modulul de colectare a alarmelor este un modul informatic declarat la nivelul de intrare în sistemul informatic de monitorizare a alarmelor. Modulul de colectare are rolul principal de a asigura medierea între aplicațiile informatice care pilotează elementele de rețea şi modulul de prelucrare a alarmelor.

Din punct de vedere fizic, modulul de colectare a alarmelor poate fi implementat direct la nivelul staţiilor concentratoare de alarme sau la nivelul central al stației de prelucrare a alarmelor. Pentru a folosi interfața de comunicație integrată în OMNeT++ am implementat modulul la nivelul central al aplicației informatice.

Arhitectura internă a modulului de colectare a alarmelor este orientată obiect şi foloseşte biblioteca de funcţii adaptive de comunicare *ACE (Adaptive Communication Environment)* [65], [109], [110], [111] şi [135]. ACE constituie astfel nucleul modulului de colectare a alarmelor, fiind o bibliotecă de primitive pentru gestiunea comunicaţiilor şi a serviciilor asociate comunicaţiilor. Fiecare funcție oferită de modulul de colectare a alarmelor este implementată folosind unul sau mai multe servicii ACE.

Biblioteca de funcţii sau servicii ACE prezintă avantajul de a fi portabilă pe mai multe sisteme de operare şi astfel poate fi folosită pentru colectarea alarmelor de la diferite rețele de telecomunicaţii care utilizează sisteme de operare diferite.

Schema generală de implementare a modulului de colectare a alarmelor este prezentată în *Figura 6.3*:

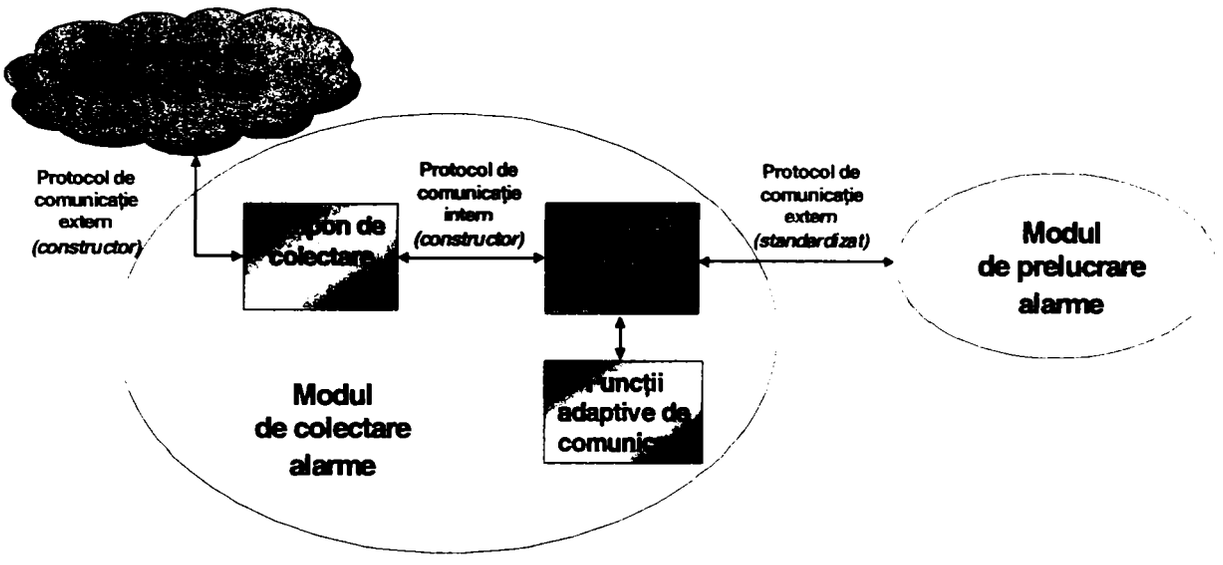


Figura 6.3 Schema de principiu a modulului de colectare a alarmelor

Tamponul de colectare din *Figura 6.3* are rolul de a consuma alarmele transmise de aplicația informatică de pilotare a elementelor din rețea. În general, protocolul de comunicație între aplicația de pilotare și tamponul de colectare este un protocol specific constructorului de echipamente. Pentru necesitatea de simulare este suficient să considerăm că tamponul de colectare este populat de evenimente având o anumită distribuție de intrare. În acest caz, aplicația de pilotare a elementelor de rețea este înlocuită de un proces de servire a unor evenimente având diferite distribuții posibile.

Dispecerul de flux din *Figura 6.3* are rolul de a adapta și negocia transferul de blocuri de alarme de la tamponul de colectare către modulul de prelucrare a alarmelor din etajul superior al aplicației informatice.

Protocolul de comunicare între dispecerul de flux și tamponul de colectare este un protocol intern specific constructorului de echipamente. Pentru a realiza implementarea în OMNeT++, deoarece am considerat că tamponul de colectare este de fapt o aplicație de servire a unor evenimente, protocolul intern folosit este un protocol generic de consumare a unei cozi de așteptare gestionată printr-un mecanism clasic de stivă.

Protocolul extern de comunicație dintre dispecerul de flux și procesul client consumator de flux de alarme poate fi orice protocol de transfer de fișiere. Am utilizat protocolul de transfer de fișiere *FTP (File Transfert Protocol).*

Dispecerul de flux are rolul de a adapta fluxul de alarme către procesul consumator de flux de alarme.

O schemă de implementare mai detaliată a modulului dispecer de flux de alarme din cadrul modulului compus de colectare a alarmelor este prezentată în *Figura 6.4*:

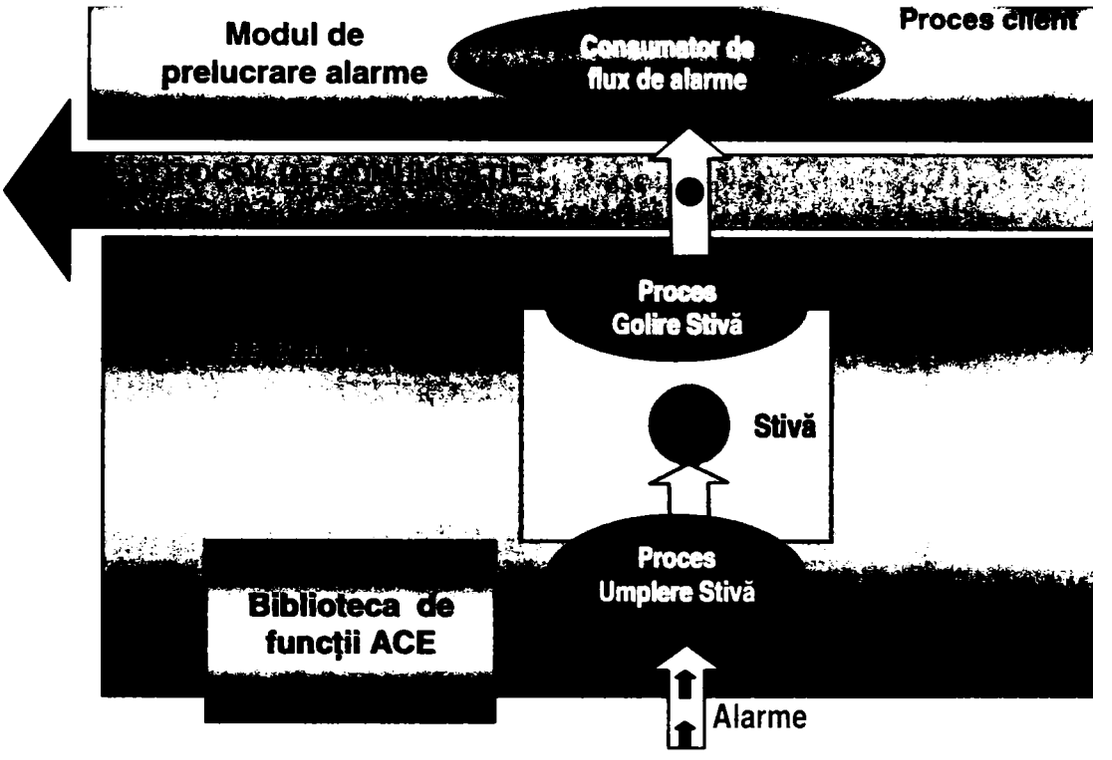


Figura 6.4 Schema de principiu a modulului dispecer de flux de alarme

Coada de așteptare din *Figura 6.4* pentru consumarea alarmelor poate fi reprezentată ca în *Figura 6.5*:

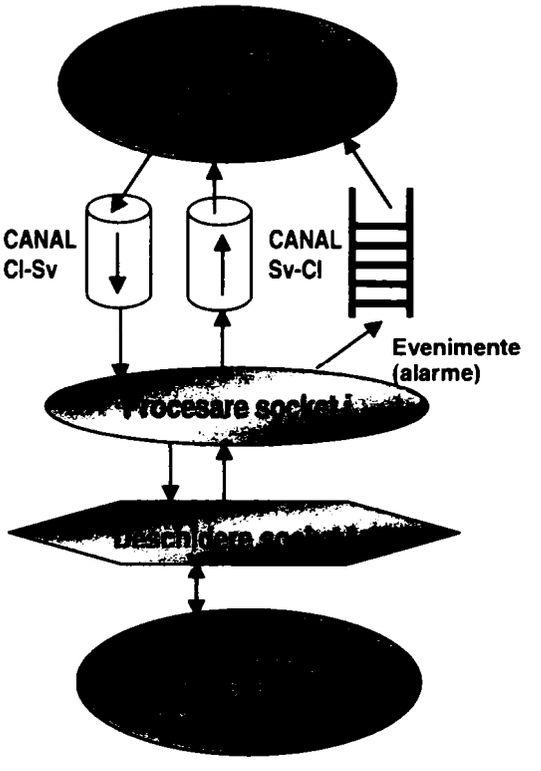


Figura 6.5 Stiva pentru colectarea alarmelor

Principiul de transmitere a blocurilor de alarme de la tamponul de colectare a alarmelor către procesul consumator de flux de alarme, prin intermediul dispecerului de flux, poate fi realizat folosind trei moduri de transfer: *permanent*, *push* și *pull*.

În modul de transfer *permanent*, blocurile de alarme sunt trimise în regim constant către consumatorul de flux de alarme, dialogul fiind continuu și sincronizat pe anumite mesaje de control, ca în *Figura 6.6*:

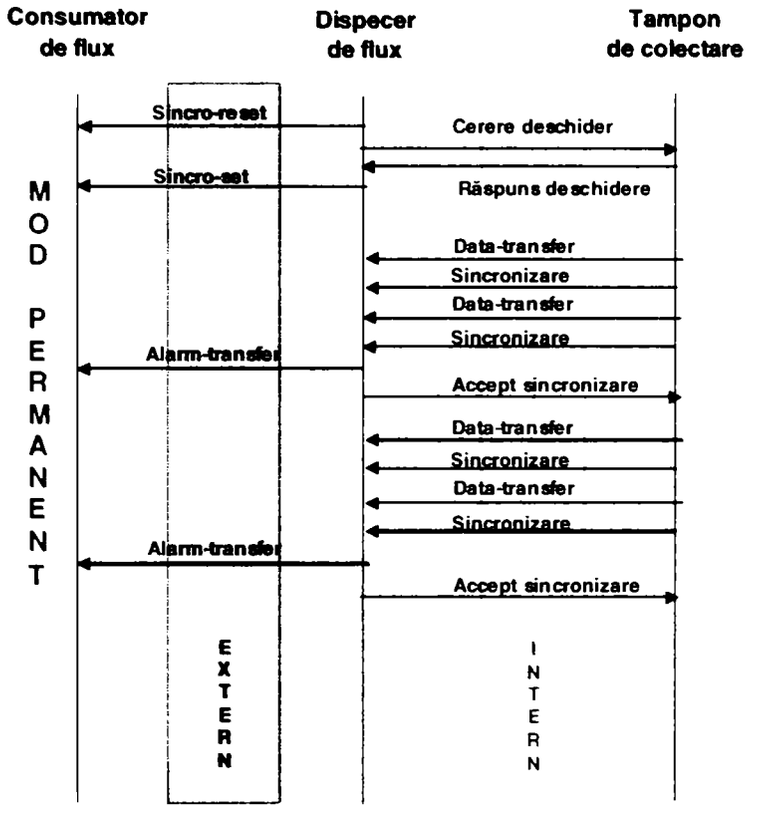


Figura 6.6 Modul de transfer *permanent*

Modurile de transfer *push* și *pull* realizează o conexiune la cerere a fluxului de alarme, fie la cererea modulului consumator de alarme (atunci când acesta consideră ca a ajuns la o limită inferioară în coada de așteptare și poate trata în continuare un alt bloc de alarme), fie la cererea tamponului de colectare (de exemplu atunci când acesta consideră că depășește o anumită limită superioară de umplere).

Modul de transfer *push* realizează o extragere sincronizată de blocuri de alarme la cererea modulului colector de alarme, așa cum este reprezentat în *Figura 6.7*:

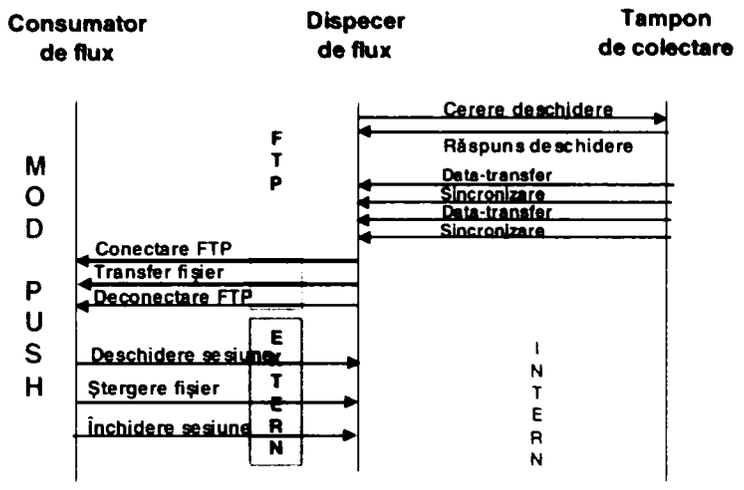


Figura 6.7 Modul de transfer *push*

Modul de transfer *pull* reprezintă, de asemenea, o extragere sincronizată de blocuri de alarme la cererea modulului de analiză a alarmelor, așa cum este reprezentat în *Figura 6.8*:

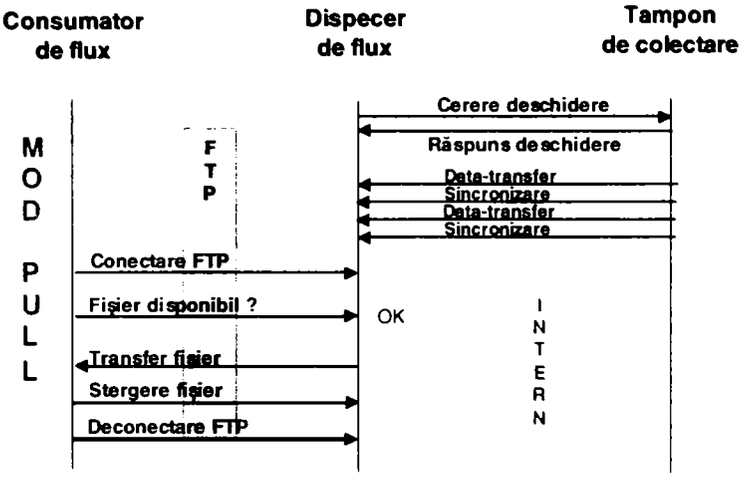


Figura 6.8 Modul de transfer *pull*

Pentru a putea folosi protocolul standard *FTP (File Transfer Protocol)* în dialogul dintre consumatorul de flux implementat în mediul de dezvoltare OMNeT++ și dispecerul de flux implementat pe platforma rețelei de telecomunicații, am ales modul de transfer *pull* pentru colectarea alarmelor. Utilizarea modurilor de transfer *permanent* și *push* necesită introducerea unui protocol extern de comunicare între procesul client consumator de flux și procesul server dispecer de flux.

6.5 Modulele pentru analiza alarmelor

În continuare se prezintă schema de implementare software a modulelor informatice pentru analiza alarmelor, detaliată în *Figura 6.9*:

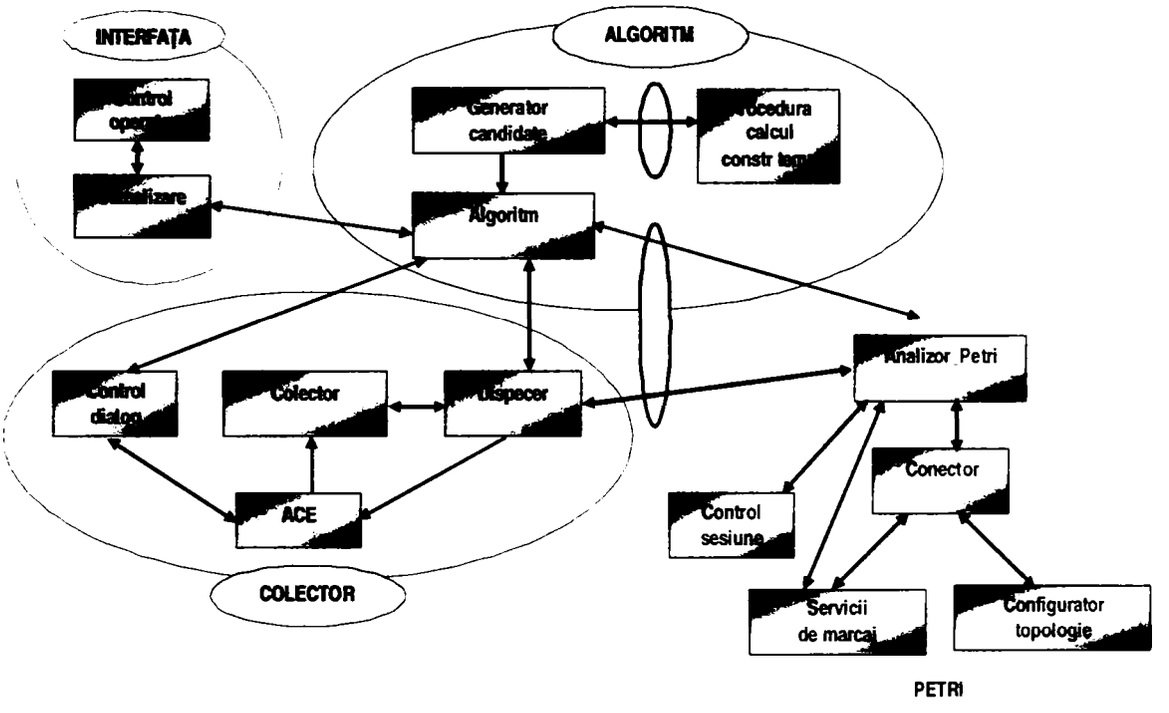


Figura 6.9 Schema de implementare a modulelor informatice

După cum am arătat în paragraful precedent, modulul informatic *Colector* conține un tampon de colectare a alarmelor care este dedicat serviciului activ și încapsulează serviciile adaptive de comunicații.

Modulul informatic *Algoritm* conține elemente de supervizare a sesiunilor de comunicație și de deschidere și închidere a socket-urilor asociate serviciilor și gestionează evenimentele care ajung în socket-uri.

Modulul informatic *Petri* construiește și analizează rețeaua Petri etichetată de cronicile frecvente descoperite în urma aplicării algoritmului.

Modulul informatic *Interfața* oferă facilități de vizualizare a soluțiilor determinate ca fiind cronici frecvente în urma aplicării algoritmului, soluții consolidate sau nu de către modulul de analiză a rețelei Petri.

În *Figura 6.9* am indicat prin intermediul unor elipse locurile unde se va întrerupe dialogul între modulele informatice pentru a realiza scenariile de testare a configurației, care vor fi prezentate în paragraful §6.9.

În *Figura 6.10* se prezintă structura mesajelor utilizate pentru transferul de blocuri de alarme :

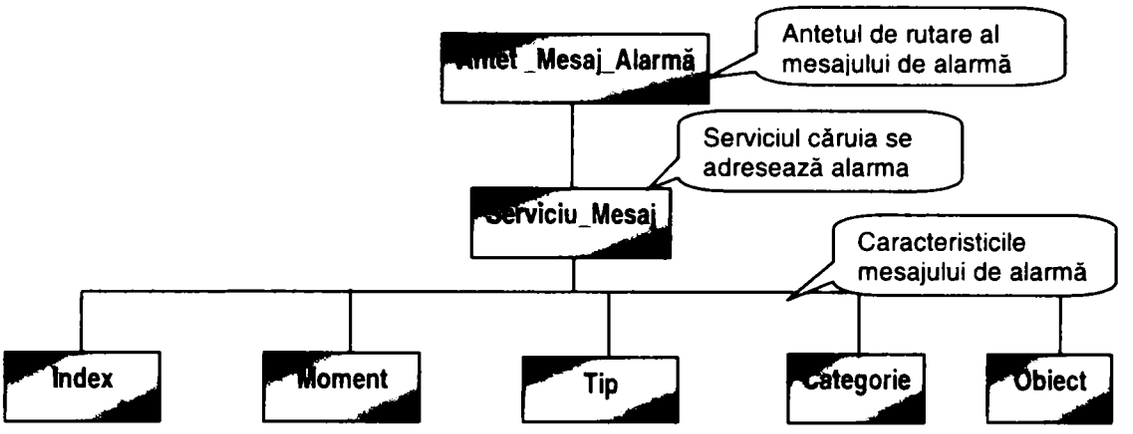


Figura 6.10 Structura mesajelor de transport de alarme

Pentru analiza alarmelor în modulele informatice am utilizat o parte din caracteristicile alarmelor, care sunt încapsulate în mesaje de alarmă. Astfel, un mesaj de alarmă este constituit dintr-un antet pentru transportarea mesajului între modulele informatice, un indicator de serviciu și caracteristicile principale ale unei alarme (index, moment de apariție, tip, categorie, obiectul la care se referă).

Clasele definite în limbajul C++ pentru implementarea funcțiilor de analiză a alarmelor sunt prezentate în *Figura 6.11*:

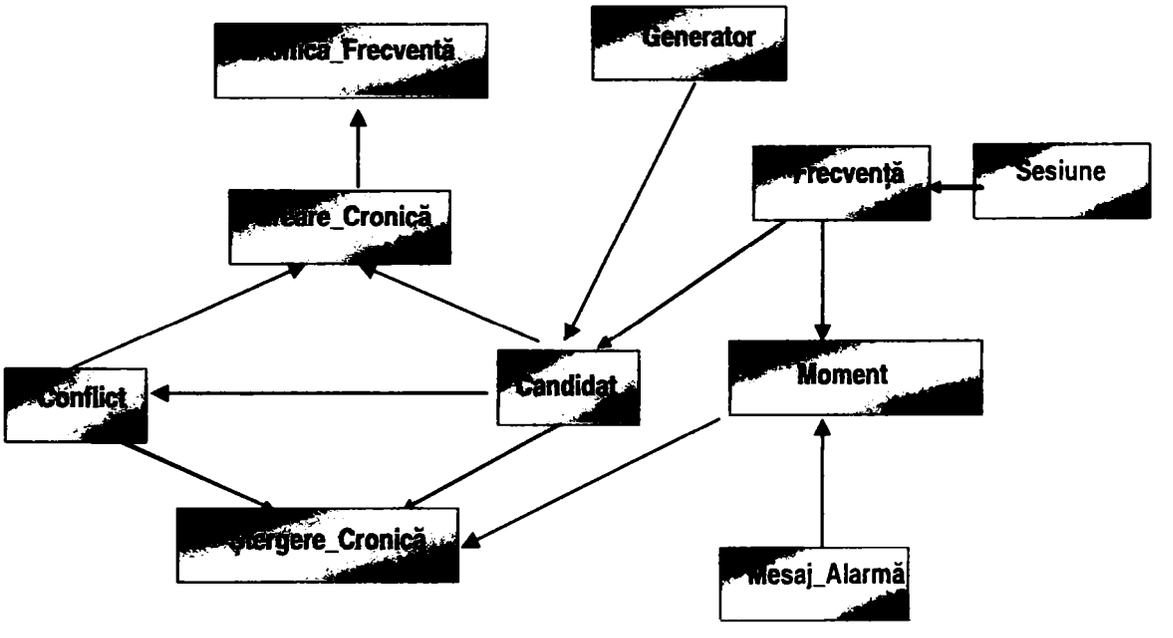


Figura 6.11 Clasele C++ pentru analiza alarmelor

- Clasa *Moment* implementează lista de momente de apariție a evenimentelor alarme și servește procedurii de calcul a constrângerilor temporale dintre alarme,
- Clasa *Frecvență* realizează frecvența de apariție a unei alarme pentru o fereastră de observație parametrabilă și folosește procedurii de calcul a frecvențelor de apariție a cronicilor candidate.

- Clasa *Candidat* generează cronicile candidate de anumite dimensiuni pentru aplicarea algoritmului.
- Clasa *Ştergere_Cronică* efectuează simplificarea arborelui de generare a cronicilor, fie în urma calculului frecvenţelor, fie în urma analizei situaţiilor din reţeaua Petri.
- Clasa *Creare_Cronică* este o reprezentare internă a unei succesiuni de alarme pentru aplicarea algoritmului
- Clasa *Conflict* afectează şi supervizează tranziţiile reţelei Petri, având acces la resursele partajate între mai multe cronici frecvente.
- Clasa *Cronică_frecventă* recepţionează mesajele soluţii finale ale algoritmului şi le transmite clienţilor destinatari de vizualizare a soluţiilor.

În plus faţă de aceste clase principale, folosite de către modulele informatice, am definit şi utilizat anumite servicii pentru gestionarea alarmelor:

- *Alarm_Data_Flux* este un serviciu de flux de date, generic pentru evenimentele din reţea. Rolul principal al acestui serviciu este de a transmite fluxul de date către un proces client sau de a efectua o scriere pentru memorarea alarmelor într-o zonă tampon, în vederea constituirii jurnalului de alarme.
- *Alarm_Data_Management* este un serviciu de abonament la un flux de date, utilizat pentru a transmite ordinul de emitere a fluxului de date către un proces client. Abonamentul propriu-zis se realizează în prin serviciul *Alarm_Data_Subscription*.
- *Alarm_Data_Subscription* este un serviciu de gestiune a zonelor tampon de alarme, având rolul de a permite unui proces client exterior de a şterge colecţiile de alarme din *Alarm_Data_Flux*.

Clasele serviciului *Alarm_Data_Flux* pot fi sintetizate în trei categorii: clase de construcţie, clase de funcţionare şi clase de infrastructură.

Clasele de construcţie ale serviciului *Alarm_Data_Flux* sunt următoarele:

- *Alarm_Service* este un serviciu generic ce poate fi instanţiat de mai multe ori,
- *Alarm_Service_Task* cere executarea unui serviciu *Alarm_Data_Flux* care creează obiectele necesare funcţionării acestui serviciu.

Clasele de funcţionare sunt următoarele:

- *Alarm_Session* pentru a emite mesajele către infrastructură,
- *Alarm_Session_Analyzer* pentru a ordona mesajele provenite de la modulele de analiză ale alarmelor către *Alarm_Session*.

Clasele de infrastructură sunt:

- *Alarm_Client* pentru a emite mesaje către procesele client care le vor consuma,
- *Alarm_Client_Analyzer* pentru gestionarea listei de procese client.

Clasele serviciului *Alarm_Data_Management* sunt următoarele:

- *Alarm_Management_Service* pentru inițializarea serviciului prin declararea interfeței cu modulul de control al algoritmului,
- *Alarm_Data_Impementationl* pentru recepția cererilor de abonament de tipul *Alarm_Data_Subscription.*

Clasele *Alarm_Data_Subscription* sunt următoarele:

- *Alarm_Data_Subscription_Service* pentru inițializarea serviciului de abonament la un flux de mesaje de alarme primite de la modulul colector de alarme,
- *Alarm_Data_Subscription_Implementation* pentru operațiile de gestiune a zonei tampon a alarmelor.

Stabilirea conexiunii modulelor de analiză a alarmelor se realizează conform următorului schimb de mesaje care duce la inițializarea algoritmului și deschiderea primei sesiuni de colectare a alarmelor, așa cum este reprezentat în *Figura 6.12*:

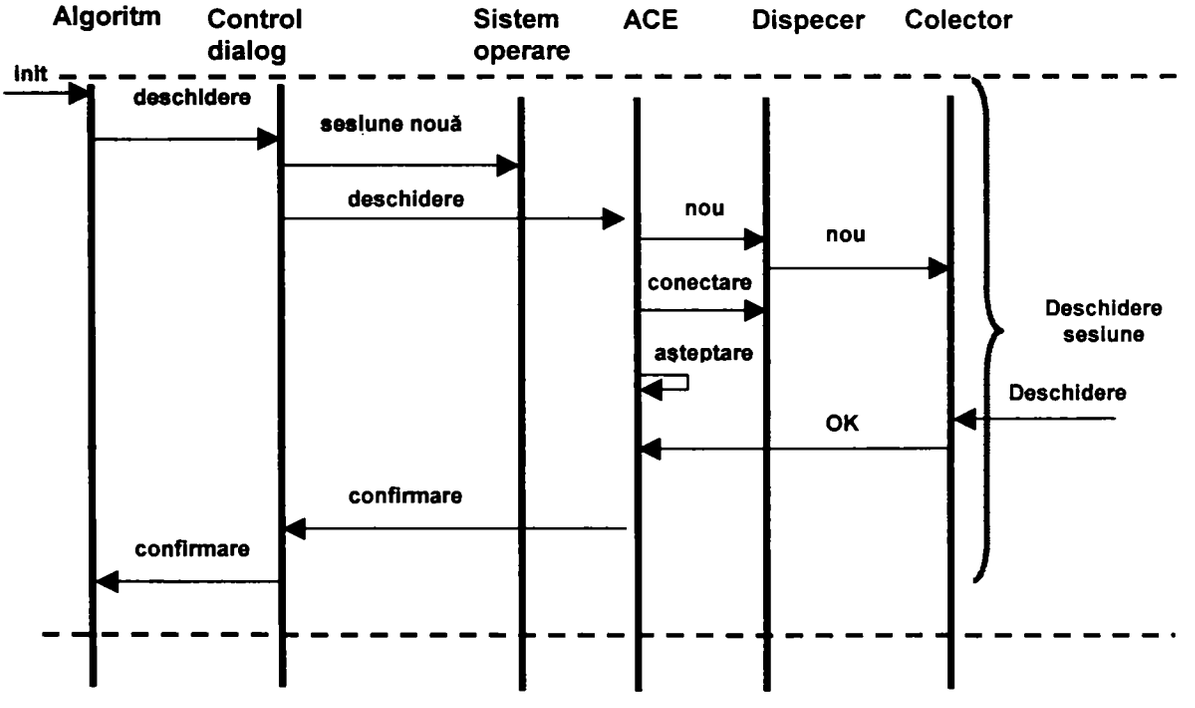


Figura 6.12 Diagrama mesajelor de inițializare

Pentru achiziția și sincronizarea alarmelor se realizează schimbul de mesaje din *Figura 6.13*:

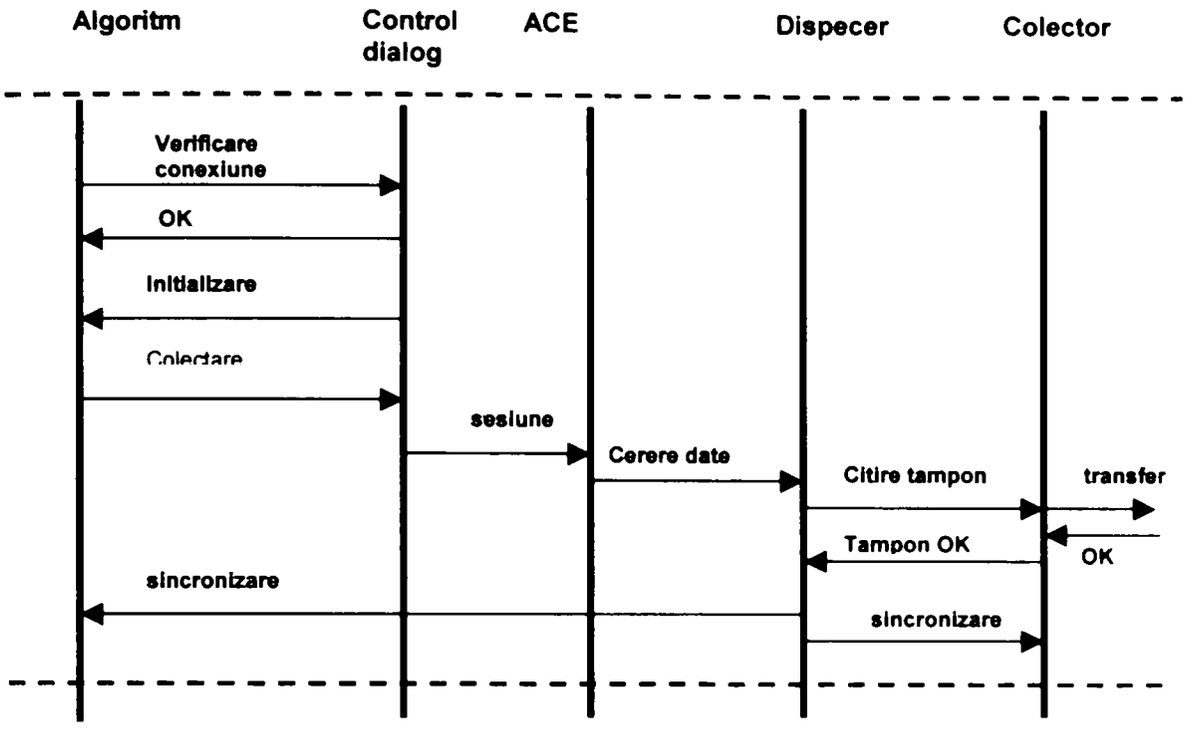


Figura 6.13 Diagrama mesajelor pentru colectarea alarmelor

Pentru analiza cronicilor de alarme se apelează, în paralel, modulul de prelucrare a alarmelor şi modulul de asamblare şi analiză a reţelei Petri, conform schimbului de mesaje din *Figura 6.14*:

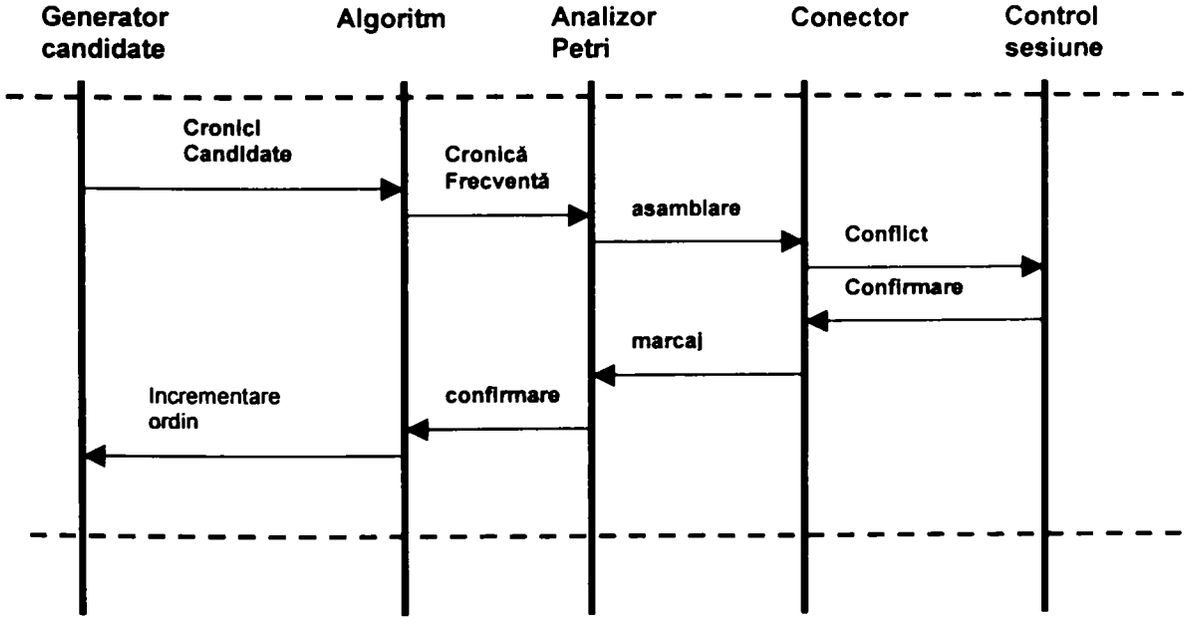


Figura 6.14 Diagrama mesajelor de transmitere a blocurilor de alarme

6.6 Modulul de prelucrare a alarmelor

Limbajul de descriere funcţională utilizat pentru implementarea algoritmului de recunoaştere a cronicilor de alarme este bazat pe *primitive*. Primitivele sunt realizate prin funcţii scrise în limbajul de programare *C++*.

Expresiile funcţionale ale celor 4 primitive principale ale limbajului de descriere funcţională, dezvoltat pentru implementarea modulului de prelucrare a alarmelor, pot fi rezumate prin următoarele definiţii:

Definiţia 6.1

Primitiva *eveniment()* reprezintă momentul de apariţie t al unei alarme de un anumit tip a către modulul de prelucrare a alarmelor:

$$eveniment(a,t) \qquad (6.1)$$

Definiţia 6.2

Primitiva *non-eveniment()* reprezintă absenţa sau dispariţia unei alarme în intervalul de timp dintre două momente t_1 şi t_2:

$$non\text{-}eveniment(a,[t_1,t_2]) \qquad (6.2)$$

Definiţia 6.3 Primitiva *durata()* implementează menţinerea unei alarme în intervalul de timp dintre două momente t_1 şi t_2:

$$durata(a,[t_1,t_2]) \qquad (6.3)$$

Definiţia 6.4 Primitiva *contor()* implementează o variabilă de numărare a apariţiilor unei alarme a între două momente t_1 şi t_2:

$$contor(min,max,a,[t_1,t_2]) \qquad (6.4)$$

Folosind primitivele descrise mai sus, se va putea implementa orice situaţie de generare a alarmelor. Pentru a exemplifica primitivele prezentate, considerăm câteva apariţii ale unei alarme a, ca în figura *Figura 6.15:*

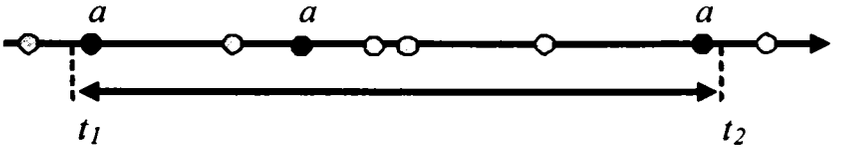


Figura 6.15 Exemplu pentru aplicarea primitivei *contor()*

În *Figura 6.15* este exprimată o alarmă *a* care apare de 3 ori, ceea ce se poate exprima cu ajutorul primitivei *contor()*, astfel:

$$contor(0,t_2,a,[t_1,t_2])=3 \quad (6.5)$$

Relațiile dintre primitivele *eveniment()* şi *non-eveniment()* în raport cu primitiva *contor()* pot fi exprimate astfel:

$$eveniment(a,t)=contor(1,\infty,a,[t,t]) \quad (6.6)$$

$$non\text{-}eveniment(a,[t_1,t_2])=contor(0,0,a,[t_1,t_2]) \quad (6.7)$$

Cu aceste notaţii, o alarmă *a* care rămâne în starea activă timp de maxim 30 de momente poate fi descrisă astfel:

```
cronica a::durata[30]
{
        eveniment (a[old,activ], t1);
        non-eveniment (a[activ, ?da], (t1,t2));
        t2-t1 in [0,30];
        throw &cronica;
}
```

În mod similar, o alarmă *a* care rămâne activă până la apariţia unei alarme *b* poate fi descrisă astfel:

```
cronica a::activ[&b]
{
        eveniment (a[old-a,active], t1);
        eveniment (b[old-b,active], t2);
        non-eveniment(a[active,?da),(t1,t2));
        throw &cronica;
}
```

Un exemplu de cronică de alarme pentru 3 acţiuni de reiniţializare aparute timp de 10 momente:

```
cronica start::reinit[3,3,10]
{
        eveniment (start, t0);
        contor (3, ∞,reinit, (t0,t1));
        t1-t0 in [0,10];
        throw &cronica;
}
```

Un alt exemplu pentru recunoaşterea a minim 1 şi maxim 3 acţiuni de reiniţializare timp de 10 momente este următorul:

```
cronica start::reinit[1,3,10]
```

```
{
        eveniment (start, t0);
        contor (1,3, reinit, (t0,t1));
        t1-t0 in [0,10]
        throw &cronica;
}
```

Un alt exemplu, mai complex, exte exprimat grafic în *Figura 6.16*. Acest exemplu reprezintă o succesiune de alarme referitoare la reinițializaea unei legături care necesită reinițializare a 3 echipamente de comunicații.

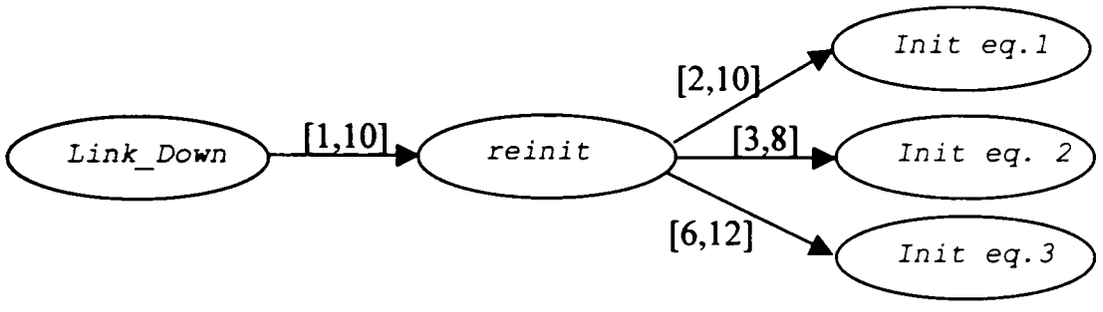


Figura 6.16 Cronică de alarme pentru reinițializarea unui echipament

Cronica de alarme din *Figura 6.16* se definește în limbajul de descriere al cronicilor de alarme astfel:

```
cronica Link_Down::reinit[3,3,&eqpt,&eqpt_1,&eqpt_2,&eqpt_3]
{
        eveniment (Link_Down[&eqpt], t1);
        eveniment (reinit[&eqpt], t2);
        t2-t1 in [1,10];
        eveniment (init[&eqpt_1], t3);
        eveniment (init[&eqpt_2], t4);
        eveniment (init[&eqpt_3], t5);
        t3-t2 in [2,10];
        t4-t2 in [3,8];
        t5-t2 in [6,12];
        throw &cronica;
}
```

Programul, dezvoltat în mediul integrat OMNeT++, are următorul mod de funcționare: pornind de la analiza unui jurnal de alarme se generează colecții de secvențe de cronici candidate și se construiește o rețea Petri pentru generarea alarmelor. Cronicile de alarme rezultate vor fi considerate soluții dacă respectă anumite condiții, cum ar fi un număr minim de apariții și ordinul maxim detectat.

Recunoașterea cronicilor se face în două etape. În prima etapă se determină cronicile în spațiul paralel, unde cronicile sunt caracterizate doar prin tipul de alarme,

fără constrângeri temporale. Se reţin ca soluţii doar acele cronici care au un număr de instanţe mai mare decât frecvenţa de apariţie minimă considerată. În cea de-a doua etapă se construieşte o reţea Petri având etichetate tranziţiile cu cronicile obţinute ca soluţii în etapa precedentă. Procedura este iterativă, la fiecare iteraţie ordinul cronicilor (numărul de alarme componente) creşte până când nu se mai găsesc instanţe de cronici care să permită determinarea unei soluţii.

Procedurile şi biblioteca de funcţii dezvoltate în limbajul de programare C++ sunt apelate de mediul de dezvoltare OMNeT++ în momentul asamblării reţelei Petri pentru simularea generării alarmelor.

Obiectivul major al modulului de integrare pentru asamblarea reţelei Petri este de a identifica informaţiile relevante în raport cu alarmele analizate.

Modulul care implementează generarea de cronici candidate, pe baza procedurilor în spaţiul paralel şi în spaţiul serial, este implementat, de asemenea, într-o bibliotecă de funcţii. Această bibliotecă este alimentată de către evenimentele din jurnalul de alarme analizat, care sunt alarmele. Modulul care conţine biblioteca de proceduri şi funcţii de generare a cronicilor candidate acţionează ca un server, furnizând rezultate către etapa ulterioară a asamblării reţelei Petri.

Task-urile care pot fi executate de către modulul central, la un moment dat, sunt următoarele:

- Citirea stării blocurilor de alarme prin mecanism de polling către clienţi,
- Memorarea stării curente a blocurilor de alarme,
- Emiterea de mesaje de alarme către blocul de operare, în urma detecţiei unei tranziţii a stării blocurilor de alarme în raport cu starea memorată anterior,
- Retransmiterea alarmei curente, în urma unei cereri directe,
- Retransmiterea telecomenzilor către sistemul de supervizare,
- Detecţia telecomenzilor transmise pe magistrală şi retransmiterea lor către sistemul de supervizare,
- Detecţia şi semnalarea alarmelor către modulul de analiză.

6.7 Modulul de asamblare a reţelei Petri

Pornind de la teoria reţelelor Petri, prezentată în capitolul anterior, în acest capitol vom descrie generarea alarmelor în reţelele de telecomunicaţii folosind reţelele Petri.

Reţelele Petri cu semantică de ordine parţială pot reprezenta evenimente concurente, adică paralele, independente între ele. Această caracteristică este importantă şi foarte bine adaptată pentru sisteme distribuite în spaţii mari care sunt în general constituite din elemente de reţea care evoluează independent în timp, având propriul lor ceas intern de evoluţie şi care se sincronizează în reţea pe anumite evenimente comune.

Pentru aplicarea rețelelor Petri la aspecte ale propagării erorilor, vom folosi un model matematic complet în care fiecare poziție în rețeaua Petri reprezintă o anume alarmă sau o situație de eroare descrisă de cronica de alarme recunoscută în prealabil. Prezența sau absența unui jeton dintr-un loc semnifică dacă o alarmă este activă sau nu. În acest caz este evident ca vom utiliza rețele Petri de capacitate unitară. Acest lucru este echivalent cu a spune că sunt rețele Petri sigure. În aceste rețele Petri sigure fiecare loc poate conține cel mult un singur jeton.

Distribuția jetoanelor peste locurile curente ale rețelei Petri constituie marcajul rețelei. Marcajul este denumit de asemenea starea globală a rețelei Petri. Distribuția jetoanelor asupra unui subset de locuri curente ale rețelei Petri constituie astfel un sub-marcaj. Sub-marcajele mai sunt denumite și stări locale.

Fiecare tranziție este asociată cu o schimbare în starea de eroare pe care o vom numi propagarea erorilor. Execuția unei tranziții schimbă stările locale și de asemenea produce alarma asociată dintr-o mulțime predefinită de alarme.

Problema de diagnosticare și de monitorizare constă în analiza evoluției în timp a stărilor pentru o anumită mulțime de alarme observate. În acest caz, stările nu mai reprezintă erori sau alarme, ele reprezintă stări de operare ale rețelei care este monitorizată. Deoarece aceste stări de operare pot fi reprezentate de stări ale rețelelor Petri, formalismul acestora este bine adaptat pentru descrierea problemei.

Semantica de ordin parțial a rețelelor Petri este bazată pe relațiile cauzale dintre diferitele execuții de tranziții sau evenimente. Astfel, în secvența ordonată de execuții $S=T_1,...T_n$, oricare tranziție T_j utilizând o resursă sau un jeton setat anterior de o tranziție T_i, va apărea ca o consecință directă a lui T_i. Această proprietate definește graful de cauzalitate notat *CG(S)*.

Pentru construcția grafului de cauzalitate, se definesc componentele elementare triplete (m_i,T,m_j), unde m_i este cel mai mic sub-marcaj care permite tranziția t și m_j este sub-marcajul care rezultă în urma execuției tranziției.

Să presupunem un marcaj inițial ca în *Figura 6.17* :

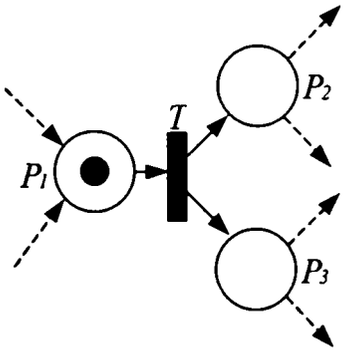


Figura 6.17 Exemplul unui marcaj inițial

Printr-o proprietate de paralelism sau concurenţă se poate ajunge în marcajul final reprezentat în *Figura 6.18* :

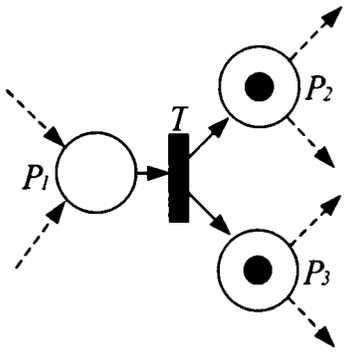


Figura 6.18 Exemplul unui marcaj final

Marcajul iniţial şi marcajul final din cele două figuri de mai sus pot fi exprimate sub forma unor componente elementare, ca în *Figura 6.17* :

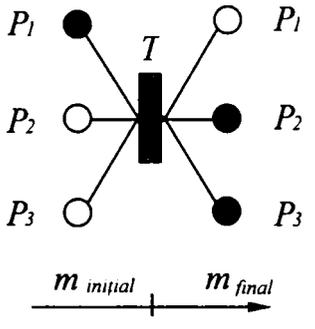


Figura 6.19 Dinamica marcajului reţelei Petri

Aceste componente elementare pot fi interconectate într-un ansamblu denumit macro-componente. Macro-componentele sunt reprezentate în mod uzual prin forma comprimată a pieselor unde se indică numai sub-marcajele.

Macro-componentele sunt interesante atunci când sistemul analizează posibile pierderi de alarme deoarece în acest caz este mai convenabil să se reprezinte propagarea erorilor prin concatenarea macro-componentelor. O singură tranziţie este vizibilă în acest caz şi înlocuieşte o pierdere de alarme.

6.7.1 Propagarea jetoanelor

Anumite reguli trebuie respectate pentru a putea asambla macro-componentele. Aceste reguli sunt studiate în lucrarea [6], unde se demonstrează de asemenea că macro-componentele pot fi reduse la minim prin reprezentarea unui graf unic de cauzalitate

determinat de tranziţiile validate şi executate. Astfel, propagarea jetoanelor în reţelele Petri reprezintă de fapt propagarea cronicilor de alarme în sistemul supervizat.

Propagarea jetoanelor poate fi reprezentată grafic sub forma unui ansamblu de componente elementare, ca în *Figura 6.20*:

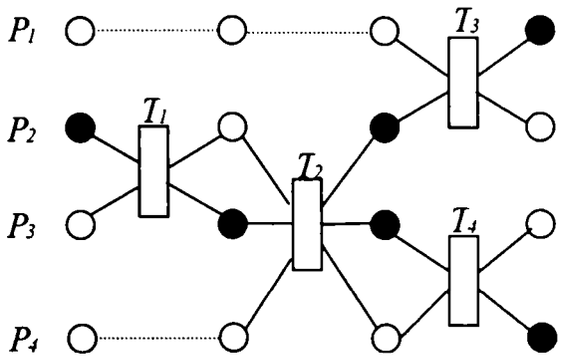


Figura 6.20 Propagarea jetoanelor

Pentru simplificarea reprezentării, ansamblul din *Figura 6.20* poate fi exprimat numai prin conectarea componentelor elementare într-un graf de cauzalitate, prezentat în *Figura 6.21*:

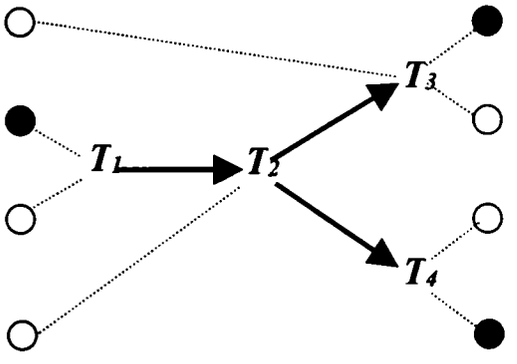


Figura 6.21 Exemplu de graf de cauzalitate

De exemplu, pentru algoritmul de recunoaştere a cronicilor frecvente prezentat în paragraful §2.5, un detaliu al reţelei privind soluţiile posibile pornind de la cronica *a* prin asamblare serială, se reprezintă astfel:

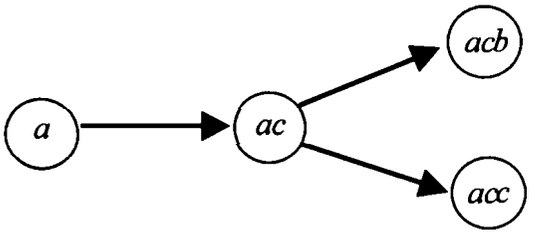


Figura 6.22 Exemplu de asamblare serială

Problema de asamblare a reţelei Petri constă în determinarea traiectoriei propagării cronicilor frecvente determinate de către algoritm asupra mulţimii de alarme analizate. În procesul de diagnosticare, fiecare cronică de alarme trebuie asociată unei execuţii ale unei tranziţii în reţeaua Petri.

Pentru a putea introduce evenimente aleatorii în descrierea problemei de supervizare, am utilizat un model probabilistic. Evenimentele aleatorii din acest model probabilistic pot fi următoarele :

- pierderea unor alarme,
- frecvenţa relativ redusă a anumitor erori spontane,
- fiabilitatea măsurărilor [66] din elementele de reţea.

Modelul probabilistic permite discriminarea între diferitele evoluţii de stări care generează alarmele. Modelul trebuie să păstreze concurenţa şi ordinea parţial semantică a reţelelor Petri. Sistemul de supervizare care colectează şi analizează alarmele în mod secvenţial va construi mai multe soluţii parţiale, pe care le va analiza.

Presupunem că este posibil să recuperăm o secvenţă de alarme dacă aceasta este compatibilă cu ordinea parţială a erorilor care le-au produs. Această ipoteză de lucru se numeşte *ipoteza observaţiilor cauzale*. Se pot face ipoteze de lucru mai puternice, de exemplu că alarmele sunt observate în aceeaşi ordine în care au fost produse. Această ipoteză nu este însă respectată în sistemele reale de telecomunicaţii deoarece trebuie luate în considerare întârzierile de propagare a informaţiilor în reţea.

Fiind dată o secvenţă de alarme, sistemul de monitorizare conectează componentele elementare şi determină un ansamblu care descrie observaţia.

Deoarece reţelele Petri nu sunt deterministe, pot rezulta mai multe ansambluri care respectă secvenţa de alarme observate. De exemplu, mai multe componente elementare pot corespunde aceleiaşi alarme, sau marcajul iniţial poate conduce la mai multe variante de traiectorii. Pentru a putea utiliza mai multe ansambluri trebuie să folosim metode de programare dinamică bazate pe calculul recursiv al grafului de cauzalitate.

Ideea de bază este că oricare două secvenţe S_1 şi S_2 care conduc la acelaşi sub-marcaj m, care generează aceeaşi secvenţă de alarme, sunt echivalente în relaţiile de dezvoltare ale unor traiectorii viitoare. Sub-marcajele m conţin toate informaţiile necesare pentru a decide conectivitatea secvenţelor de execuţie S la S_1 sau S la S_2. De aceea, dacă într-o anumită situaţie de observaţii secvenţa S_1 este mai probabilă decât secvenţa S_2 atunci S_2 poate fi eliminată din rândul candidatelor ca potenţială soluţie optimă.

Acest fapt permite introducerea noţiunii de programare dinamică pentru sub-marcajul *m*. La fiecare pas se salvează argumentul de probabilitate şi este utilizat pentru a găsi recursiv cea mai probabilă propagare a cronicilor.

Menţionăm aici că în algoritmul folosit stările sunt reprezentate de sub-marcaje şi nu de marcaje globale, ceea ce este o consecinţă importantă a formalismului de randomizare, conform lucrării [6].

În concluzie, chiar dacă algoritmul prezentat aici selectează cea mai probabilă propagare a cronicilor, se pot calcula toate soluţiile posibile. Dacă dorim să calculăm toate soluţiile posibile atunci în mod evident nu mai este necesară informaţia de probabilitate.

Principala caracteristică a metodei de asamblare cu reţelelor Petri este că rezultatul obţinut este o expresie compactă a dinamicii sistemului exprimată prin cauzalitate şi concurenţă. Ideea de bază este că se pot determina toate soluţiile dar volumul de calcul este foarte mare şi sunt necesare multe date. Prin randomizare sau aleatorizare se ia în considerare o distribuţie probabilistică în scopul de a reduce numărul de date prelucrate şi evident se reduce volumul de calcul.

O altă caracteristică foarte importantă este că procedurile algoritmului şi asamblării de reţele Petri sunt disponibile pentru monitorizarea distribuită, ceea ce este esenţial în sistemele de telecomunicaţii [26].

Menţionăm că metodele de diagnostic distribuit pot fi descrise de asemenea folosind formalismul reţelelor Petri. Pentru detalii despre metodele de diagnostic distribuit pot fi consultate lucrările [7], [33] şi [34].

6.7.2 Reţele Petri etichetate de cronici

În paragraful precedent am prezentat reţele Petri unde tranziţiile sunt etichetate de alarme şi propagarea jetoanelor în reţeaua Petri generează o ordine parţială de alarme. Sistemul de supervizare va observa secvenţele de alarme care sunt compatibile cu ordinea parţială considerată.

Reţelele Petri etichetate de alarme pot fi dezvoltate astfel încât tranziţiile să nu corespundă numai unor alarme şi să corespundă unor cronici de alarme recunoscute în prealabil. Astfel, trecerea unei tranziţii va genera alarmele corespunzătoare cronicii având constrângerile temporale definite de acea cronică. Trecerea a două tranziţii consecutive va genera o secvenţă de instanţe de cronici. Trecerea a două tranziţii concurente poate genera cronici de ordin superior. Cu acest raţionament, se poate determina că propagarea

jetoanelor în rețelele Petri va genera o secvență de alarme observate care este compusă din mai multe cronici corespunzătoare acestor alarme.

Algoritmul de recunoaştere a cronicilor calculează alarmele ordonate în timp şi le regrupează în cronici de alarme. Aceste cronici de alarme sunt trimise, în ordinea în care ele sunt recunoscute, către modulul de asamblare a rețelei Petri.

Deoarece modulul de recunoaştere a cronicilor nu determină nici o relație prealabilă între cronicile recunoscute, procesul de recunoaştere poate lucra cu mai multe partiții ale jurnalului de alarme analizat.

Anumite instanțe de cronici care nu se regăsesc în toate partițiile jurnalului pot fi ignorate datorită faptului că nu se poate determina o traiectorie Petri pentru a explica alarmele observate. În acelaşi timp, prin construcție, se determină că orice scenariu de generare de alarme care este valid poate genera o partiție a jurnalului de alarme.

Prin construcția rețelei Petri, se determină că scenariile de generare a alarmelor generează fiecare câte o partiție. Astfel, procesul de supervizare poate lucra cu un spațiu restrâns la evoluția în timp a partiției alarmelor componente ale instanțelor de cronici. Acest lucru este realizat de modulul de asamblare a rețelei Petri.

Procesul de diagnosticare este bazat pe o cooperare între modulul de recunoaştere a cronicilor şi modulul de asamblare a rețelei Petri. Fluxul de intrare în modulul de recunoaştere a cronicilor este constituit de jurnalul de alarme. Fluxul de intrare în modulul de asamblare a rețelei Petri este constituit în mod exclusiv din cronicile de alarme determinate de primul modul. Aceste cronici etichetează tranzițiile rețelei Petri construite.

În continuare se prezintă un exemplu de asamblare a unei rețele Petri pentru un jurnal de alarme J care conține alarmele a,b,c şi d, unde sunt recunoscute următoarele cronici de alarme:

$$\overline{A}(J)=\begin{cases} T_1:(a,b,c,c) \\ T_2:(a,b,c) \\ T_3:(a,c,c) \end{cases} \qquad (6.8)$$

În figura următoare prezentăm rețeaua Petri care are etichetate tranzițiile cu cronicile deja recunoscute. De exemplu, cronicile de alarme *(a,b,c,c)* şi *(a,b,c)* sunt asociate tranzițiilor T_1 respectiv T_2:

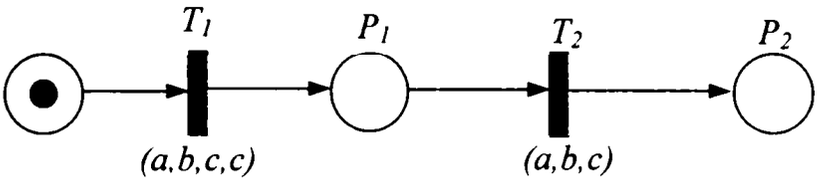


Figure 6.23 Exemplu de reţea Petri etichetată de cronici

Atunci când sunt recunoscute cronicile, de exemplu pentru cele din relaţia ***(6.8)***, aceste cronici sunt transmise modulului de asamblare a reţelei Petri. Cronicile sunt recunoscute în ordinea temporală prezentată mai sus. Un exemplu de tranziţii concurente etichetate de cronici, conform relaţiei ***(4.1)***, este prezentat în *Figura 6.24*:

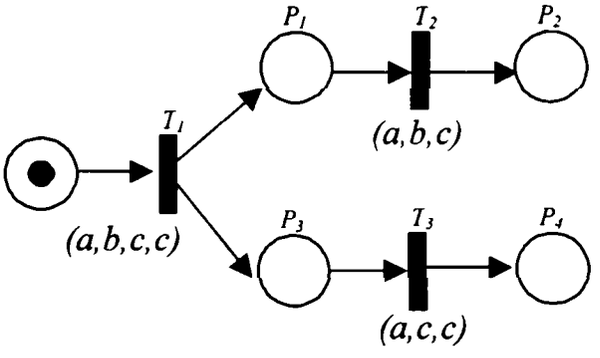


Figura 6.24 Tranziţii concurente

În figura de mai sus sunt reprezentate tranziţiile T_1, T_2 şi T_3 care sunt etichetate de cronicile de alarme deja recunoscute. Dacă urmărim asamblarea reţelelor Petri vom putea determina configuraţia dată.

Pentru început reţeaua Petri nu are nici o istorie şi deci se poate presupune că sistemul nu are nici o stare iniţială determinată. Iniţial atât algoritmul de recunoaştere a cronicilor cât şi cel de asamblare a reţelei Petri vor recepţiona prima alarmă *a*, care este memorizată în jurnalul de alarme. În acest moment nu există încă proces de recunoaştere a cronicilor. Apoi sistemul recepţionează alarmele *b* şi *c* şi atunci algoritmul de recunoaştere a cronicilor determină cronica *(a,b,c)*. Modulul de asamblare a reţelei Petri construieşte tranziţia T_2. Apoi apare încă o dată alarma *c* şi algoritmul de recunoaştere a cronicilor determină două cronici frecvente, T_1 şi T_3.

Cronica *(a,b,c,c)* etichetează tranziţia T_1 şi în reţeaua Petri se schimbă starea sistemului de la P_1 la P_3. Consumând o alarmă *c* se ajunge în poziţia P_2, iar consumând o alarmă *b* se ajunge în poziţia P_4.

Comportamentul reţelei Petri este exprimat prin relaţia ***(6.9)***:

$$\begin{cases} \text{dacă } P_1 \text{ atunci } P_2 \\ \text{sau} \\ \text{dacă } P_3 \text{ atunci } P_4 \end{cases} \quad (6.9)$$

Noţiunea de stare a reţelei Petri permite să se elimine anumite ipoteze de lucru, atunci când se ajunge în aceeaşi stare. Informaţiile conţinute în modelul de cronici sunt folosite pentru a elimina ipoteze de lucru care nu pot fi completate niciodată de o nouă instanţă de apariţie.

Orice stare finală reprezintă cel mai bun scenariu care generează o partiţie de tipar de alarme frecvente, folosind un algoritm iterativ.

Procedura de recunoaştere a cronicilor frecvente nu determină nici o relaţie între alarme, ceea ce înseamnă că, chiar dacă jurnalul de alarme a fost împărţit în mai multe ferestre de observaţie, cronicile recunoscute pot fi trimise modulului de asamblare a reţelei Petri.

Modulul de asamblare a reţelei Petri vine să completeze procedura de recunoaştere a cronicilor frecvente descrisă în Capitolul 2. Cronicile frecvente recunoscute prin aplicarea procedurilor descrise în Capitolul 2 sunt modelate sub formă de tranziţii într-o reţea Petri. Această modelare completează descrierea comportamentului reţelelor de telecomunicaţii. Modelarea reţelelor este completă şi include aspecte particulare cum ar fi propagarea alarmelor, întârzierile din reţea şi chiar pierderea de alarme.

Scopul modelării cu reţele Petri este să completeze recunoaşterea cronicilor. Ideea principală este că alarmele care au fost deja recunoscute ca fiind componente ale unor cronici să fie folosite pentru a elimina ipotezele care nu funcţionează din reţelele Petri. În mod similar, execuţia reţelei Petri poate indica, atunci când se ajunge la oprirea reţelei, că se pot elimina din calcul cronicile candidate deja recunoscute. Reţeaua Petri permite de asemenea determinarea unei mulţimi minime de alarme pierdute, necesare reconstruirii unor cronici frecvente.

Rezultatul modului Petri este o reţea Petri având tranziţiile etichetate de cronicile candidate care pot intra în două posibile interacţiuni : consecutive sau concurente. Interacţiunea consecutivă implică două cronici relaţionate cauzal şi ordonate una faţă de cealaltă. Interacţiunea concurentă înseamnă că două cronici pot rezulta într-un proces de întrepătrundere. Interacţiunea concurentă de cronici poate fi extinsă pentru a permite exprimarea unor dependinţe mai complexe între cronicile recunoscute.

6.8 Teste unitare

Verificarea unui proiect, după faza de programare, conform metodelor de management de proiecte informatice, se face prin realizarea de testelor unitare. Testele unitare efectuate, pentru verificarea funcţionării modulului de analiză a alarmelor, în vederea eliminării eventualele erori de programare, trebuie să acopere toate funcţiile oferite. Funcţiile oferite sunt clasificate în mai multe teme de testare, care sunt apoi verificate prin metode specifice.

Funcţiile oferite de modulul de analiză a alarmelor sunt :

- Iniţializarea,
- Resincronizarea,
- Funcţionarea în regim normal,
- Funcţionarea în regim excepţional (suprasarcină, regularizarea alarmelor, etc).

Lista de teste unitare, clasificată pe temele de testare precizate mai sus, este următoarea:

- Teste unitare de iniţializare - Obiectiv: se verifică iniţializarea modulelor informatice pentru analiza alarmelor.
  - Iniţializarea modulului de colectare de alarme.
  - Recepţionarea unei comenzi corecte de la modulul de analiză.
  - Iniţializarea completă a modulului de analiză şi a modulului Petri.
  - Citirea unei zone tampon de alarme la cererea modulului de analiză.
  - Parametrarea unui fişier de configurare.
  - Reconfigurarea topologiei reţelei în mod *off-line.*
  - Generarea de evenimente de intrare în procesele client.
  - Iniţializarea blocului de audit pentru evenimentele spontane – jetoane.
  - Iniţializarea blocului de statistică - contor şi reprezentare histogramă.
  - Reconectarea unei zone tampon de alarme la cererea modulului de analiză.
  - Conectarea mai multor zone tampon de alarme (multi-colector).
  - Realizarea de puncte de debug pentru verificarea mesajelor API.
- Teste unitare de resincronizare - Obiectiv: se verifică resincronizarea sistemului informatic pe anumite evenimente.
  - Resincronizarea directă la cererea unui proces client.
  - Resincronizarea directă la cererea mai multor procese client.
  - Resincronizarea prin abonament la un proces client.
  - Resincronizarea prin abonament la mai multe procese client.
  - Recepţia mesajului de resincronizare de la un proces client.
  - Recepţia mesajului de resincronizare de la mai multe procese client.

- Teste unitare pentru verificarea funcționării în regim normal - Obiectiv: se verifică interconectarea modulelor informatice la o funcționare în regim normal.
  - Recepția unui mesaj de alarme achitat.
  - Recepția unui mesaj de alarme neachitat în timpul resincronizării.
  - Recepția unui mesaj de alarme neachitat în timpul supervizării.
  - Expirarea timpului pentru abonamentul la procesele client.
  - Recepția și ignorarea unui mesaj neimplementat.
  - Mesaje de abonare la modulul de colectare.
  - Mesaje de editare a contorilor și numărul maxim de alarme gestionate.
- Teste unitare pentru verificarea funcționării în regim excepțional - Obiectiv: se verifică funcționarea modulelor în cazul unor erori de interconectare.
  - Ignorarea unei interconectări eronate.
  - Analiza unui fișier eronat al topologiei rețelei.
  - Întreruperi de comunicare între modulele informatice.
  - Zonă de tampon de alarme inaccesibilă de către modulul colector.
  - Mecanismul de regularizare a alarmelor.
  - Probleme de desincronizare a mesajelor și expirarea timpului de așteptare.
  - Refuzarea unui mesaj de resincronizare.
  - Reinițializarea generală a blocurilor în urma unei erori grave.

Testele unitare prezentate mai sus au fost realizate în proporție de 100% și am observat și corectat o serie de erori de programare sau de concepție a anumitor module. În paragraful §6.8.1 voi prezenta problemele majore care au fost corectate în urma efectuării acestor teste unitare.

6.8.1 Aspecte ale unor corecții în urma efectuării testelor unitare

În urma testelor unitare realizate, mai ales din punct de vedere al performanței modulelor de analiză a alarmelor, am efectuat mai multe corecții asupra modulelor în cauză. În continuarea acestui paragraf se prezintă aspectele majore care au dus la corectarea unor probleme de concepție.

În primul rând, datorită faptului că procedura de generare a cronicilor candidate este iterativă, calculând numărul de instanțe de cronici candidate, dacă se determină o cronică frecventă atunci toate subcronicile ei au cel puțin aceeași frecvență. Pentru exemplificarea acestei probleme, considerăm o secvență de alarme *(a,11)(b,12)(b,13),c(15)*. Problema care se formulează, în acest caz, astfel : care dintre instanțele de alarme prezente în această secvență corespund cronicii *(a,b,c)*. Se pot alege instanțele *(a,11)(b,12)(c,15)* sau *(a,11)(b,13)(c,15)* sau amândouă. Inițial am ales ca instanțele să fie disjuncte, deci două instanțe distincte de cronici să nu conțină aceeași instanță de alarmă, dar astfel se pierdea cea de-a doua cronică. În urma testelor am

eliminat această restricție. Astfel, în procedura de generare a cronicilor candidate, vor fi numărate amândouă instanțele prezentate mai sus.

În al doilea rând, o instanță de alarmă generată cu o anumită întârziere poate influența semnificativ procesul de generare de cronici candidate. De exemplu, dacă o cronică este detectată într-o serie de momente relativ apropiate, dar are o ultimă recunoaştere relativ târzie față de seria de momente de la început, de exemplu *t=[3,5,10,14,20,27,100]*, cronica rezultată va fi recunoscută la momentele *t=[3,100]*. Pentru a obține rezultate mai bune se poate separa cronica pe două intervale, *t=[3,27]+[100]* şi se poate parametra un interval maxim pentru recunoaşterea cronicilor astfel încât să nu se ia în considerare întârzierile relativ mari ale unei cronici de alarme.

6.8.2 Scara de reprezentare şi selecția automată a detaliilor

Un aspect important pentru realizarea modulului de vizualizare a soluțiilor determinate de către algoritm este scara de reprezentare dinamică. Pentru a rezolva problema alegerii scării de reprezentare, am ales teoria dezvoltată pentru selecția automată a detaliilor unei imagini, prezentată de Lindeberg în lucrarea [76].

Menționez că am mai folosit această soluție de reprezentare dinamică în articolele [116] şi [117] referitoare la analiza performanțelor rețelelor telefonice bazate pe pachete de date.

Pentru o anumită distribuție de momente de apariție, adăugăm o dimensiune care defineşte scara de reprezentare. Această scară variază în mulțimea numerelor naturale, momentele de apariție fiind puncte temporale întregi pozitive. Spațiul scării de reprezentare poate fi văzut ca fiind o funcție de aceste momente de apariție. Pentru a utiliza informația într-o formă compactă şi nu prea voluminoasă, trebuie aleasă o scară de reprezentare relativă la cronicile de alarme determinate anterior.

În acest scop, am implementat în modulul de vizualizare o facilitate de parametrare a selecției automate a detaliilor.

6.9 Rezultate experimentale

Pentru a verifica performanțele modulelor informatice prezentate în paragrafele anterioare, am realizat mai multe comparații asupra rezultatelor experimentale obținute în mai multe configurații de test. În primul rând, am definit următoarele configurații sau scenarii de test:

- Scenariul 1 : Modulul de analiză a alarmelor conține algoritmul simplu descris în paragraful §2.5, fără a folosi procedura de calcul a constrângerilor temporale şi nici procedura de asamblare a rețelei Petri aferente.

- Scenariul 2 : Modulul de analiză conține algoritmul extins cu procedura de calcul a constrângerilor temporale, descrisă în paragraful §2.6.1.
- Scenariul 3 : Modulul de analiză conține algoritmul extins și, în paralel, se asamblează o rețea Petri etichetată de soluțiile determinate în urma execuției algoritmului. Soluțiile sunt transmise modulului Petri pe măsură ce sunt determinate.

În cazul scenariului 1 am obținut o serie de rezultate experimentale care sunt considerate rezultate inițiale pentru comparație. Am aplicat scenariul 1 asupra unui jurnal de alarme care conține 3000 de înregistrări cu 25 de tipuri de alarme. Rezultatele aplicării algoritmului sunt prezentate în *Tabelul 6.1* :

Frecvența minimă considerată	*Alarme frecvente*	*Cronici candidate*	*Cronici frecvente*	*Timp de execuție algoritm [mm:ss]*
25	25	5810	366	13:27
50	17	4900	108	03:10
100	8	2890	28	00:44
250	3	1130	11	00:28
500	1	78	0	00:04
1000	0	0	0	00:00

Tabelul 6.1 Rezultatele algoritmului pentru scenariul 1

În scenariul 2 am folosit procedura de calcul a constrângerilor temporale pentru a reduce numărul de cronici candidate care sunt generate de către algoritm. Astfel, am redus complexitatea algoritmului, având mai puține cronici candidate. Timpul câștigat în execuția algoritmului prin faptul că sunt mai puține calcule de efectuat este folosit de către procedura de calcul a constrângerilor temporale. În final, am obținut un timp de execuție mai bun cu 5-10% pentru diferite configurații de test. Pentru aceeași configurație de test cum este cea folosită la scenariul 1, în cazul scenariului 2 se obțin rezultatele din *Tabelul 6.2* :

Frecvența minimă considerată	*Alarme frecvente*	*Cronici candidate*	*Cronici frecvente*	*Timp de execuție algoritm [mm:ss]*
25	25	5630	320	12:51
50	17	3440	92	02:50
100	8	2090	28	00:42
250	3	810	11	00:25
500	1	53	0	00:04
1000	0	0	0	00:00

Tabelul 6.2 Rezultatele algoritmului pentru scenariul 2

În cel de-al 3-lea scenariu intervine procedura de asamblare a reţelei Petri. Scopul principal al acestei proceduri este de a analiza cronicile frecvente determinate anterior şi de a elimina acele cronici care sunt redundante deoarece cauzează situaţii de consecutivitate în reţeaua Petri.

Avantajul major al modulului Petri este că acesta se execută în paralel cu algoritmul extins şi nu consumă un timp suplimentar. După cum am mai arătat, pe măsură ce sunt determinate soluţii de către algoritm, aceste soluţii sunt transmise modulului Petri.

Modulul Petri asamblează o reţea Petri care are tranziţiile etichetate de cronicile frecvente recunoscute anterior şi analizează această reţea din punct de vedere a aspectelor de consecutivitate şi concurenţă. Deoarece anumite cronici frecvente sunt eliminate de acest modul, vor fi mai puţine cronici candidate şi, în final, se va ajunge la mai puţine cronici frecvente.

Trebuie menţionat aici că nu se pierd informaţii utile, deoarece cronicile frecvente eliminate sunt redundante cu celelalte cronici frecvente care au rămas în calcul. Timpul de execuţie este mai bun cu până la 10% pentru diferite configuraţii de test.

Pentru configuraţia de test folosită anterior la scenariile 1 şi 2, în cazul scenariului 3 se obţin rezultate din *Tabelul 6.3* :

Frecvenţa minimă considerată	*Alarme frecvente*	*Cronici candidate*	*Cronici frecvente*	*Timp de execuţie algoritm [mm:ss]*
25	25	5530	280	11:34
50	17	3020	70	02:29
100	8	1098	22	00:35
250	3	750	7	00:22
500	1	44	0	00:03
1000	0	0	0	00:00

Tabelul 6.3 Rezultatele algoritmului pentru scenariul 3

Sinteza rezultatelor din tabelele de mai sus, referitoare la cronicile frecvente care sunt obținute ca soluții finale, se poate reprezenta grafic ca în *Figura 6.25*:

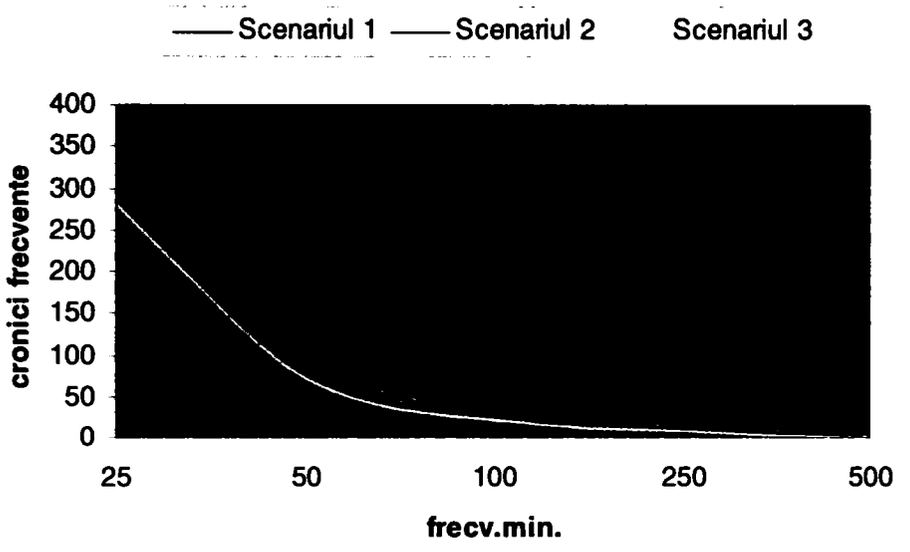


Figura 6.25 Graficul cronicilor frecvente pentru scenariile prezentate

Timpul de execuție al algoritmului pentru fiecare scenariu de test considerat poate fi reprezentat ca în *Figura 6.26*:

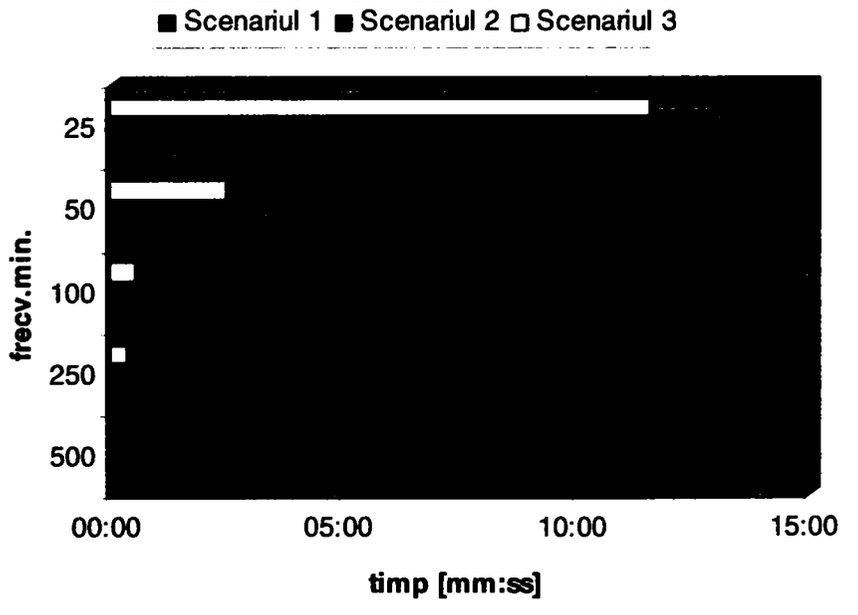


Figura 6.26 Timpul de execuție al algoritmului de recunoaștere

Se observă că se obține un timp mai bun pentru algoritmul extins și un timp și mai bun pentru algoritmul extins cu folosirea procedurii de analiză a rețelei Petri.

Comparând mai multe execuții consecutive în scenariile prezentate, pe jurnalul de alarme de 3000 de apariții cu 25 de alarme ales pentru experimentare, am ajuns la concluzia că prin aplicarea procedurii de calcul a constrângerilor temporale se îmbunătățește timpul de execuție al algoritmului, în medie, cu 8%. Pentru alte jurnale de alarme mai simple se poate obține un timp mediu mai bun cu până la 10%. Prin introducerea în paralel a unei analize a cronicilor frecvente cu rețele Petri se mai câștigă, în medie, 5% față de Scenariul 2, deci cu 13% față de Scenariul 1. Pentru jurnale de alarme mai simple se poate ajunge la un timp mediu mai bun cu până la 15%.

În *Figura 6.27* am reprezentat o sinteză a timpului mediu de execuție pentru cele 3 scenarii în cazul jurnalului de alarme folosit pentru experimentarea algoritmului:

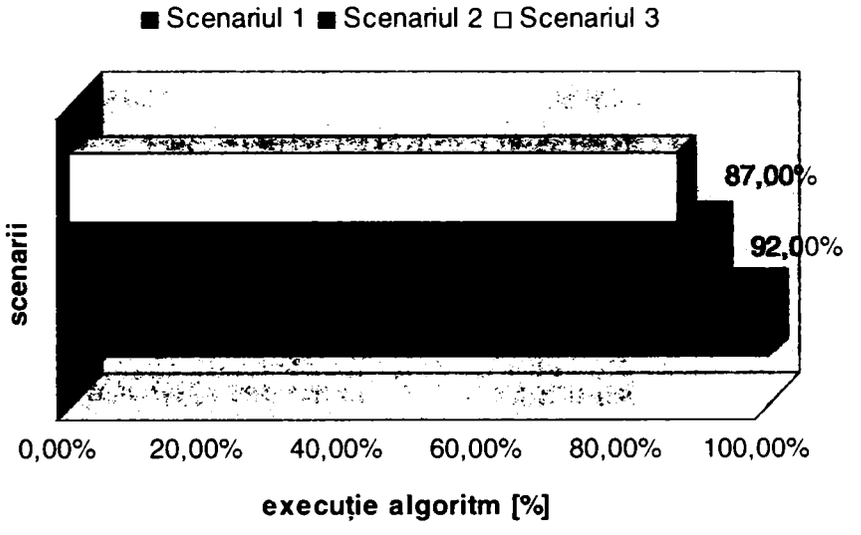


Figura 6.27 Timpul de execuție mediu pentru scenariile prezentate

6.10 Comparaţii şi concluzii

Din punct de vedere funcţional, proiectul informatic dezvoltat în OMNeT++ oferă majoritatea facilităţilor rezultate în urma proiectului profesional *FACE (Frequency Analyzer for Chronicle Extraction)* [44].

Dacă se analizează facilităţile de interfaţă cu utilizatorul, atât OMNeT++ cât şi FACE oferă o interfaţă intuitivă cu topologii de reţea care pot fi modificate dinamic, de exemplu prin adăugarea de noi module de colectare de alarme sau prin interconectarea unor module externe de calcul.

OMNeT++ are avantajul de a integra un limbaj nativ de descriere a topologiei reţelei, ceea ce conduce la o mai mare portabilitate a aplicaţiei, pe diferite platforme. FACE a fost dezvoltat pe o platformă mai puternică, dar modulele OMNeT++ pot fi recompilate şi construite pe o platformă echivalentă.

Din punct de vedere al interfeţei de ieşire, OMNeT++ poate exporta rezultatele în *plove*, FACE având un sistem proprietar de editare a rezultatelor. Din punct de vedere al colectării alarmelor, FACE este interconectat direct unei reţele de telecomunicaţii, fie în mod *off-line* (pentru achiziţia alarmelor), fie în mod *on-line* (pentru procesarea alarmelor).

Modulul de colectare de alarme din OMNeT++ ar putea oferi 3 moduri de interconectare (*permanent*, *push* şi *pull*), dar nu are implementat decât modul de extragere de pachete de alarme la cererea modulului de analiză a alarmelor.

În raport cu timpul de execuţie pentru recunoaşterea unor cronici de alarme din jurnale de alarme similare, testele efectuate arată că FACE este mai rapid în rezolvarea problemei.

Comparaţia facilităţilor celor două proiecte informatice pentru recunoaşterea cronicilor frecvente de alarme din reţelele de telecomunicaţii poate fi sintetizată ca în *Tabelul 6.4*:

Facilităţi	*OMNeT++*	*FACE*
Interfaţa utilizator	+++	+++
Modificări topologie	+++	++
Portabilitate	+++	++
Exportul rezultatelor	++	+
Interfaţa colectare	+	++
Timp de execuţie	+	+++

Tabelul 6.4 Comparaţie între OMNeT++ şi FACE

O sinteză a comparaţiei facilităţilor oferite de OMNeT++ şi FACE este prezentată în *Figura 6.28*:

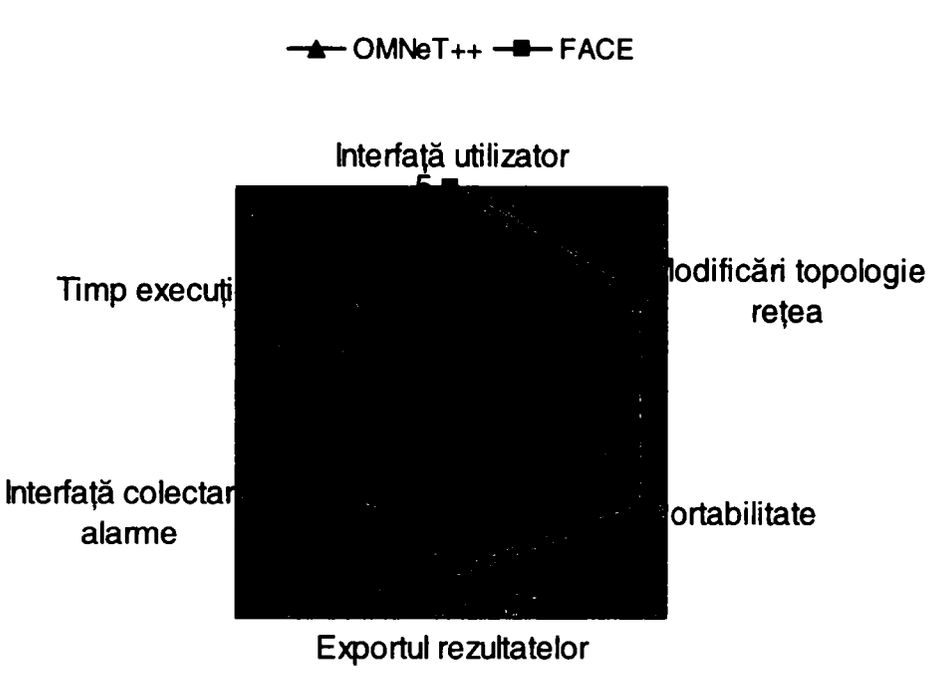


Figura 6.28 Comparația facilităților OMNeT++ în raport cu FACE

În *Figura 6.28* am notat fiecare facilitate pe o scară de la 1 la 5, cu următoarele explicații:

- Interfața cu utilizatorul – ambele proiecte prezintă o interfață grafică intuitivă cu diverse facilități specifice unor medii de dezvoltare integrate.
- Modificări de topologie – am notat mai bine OMNeT++ datorită prezenței limbajului NED.
- Portabilitate – Modulele C++ din OMNeT++ pot fi recompilate pentru mai multe platforme (*unix* sau *win32*).
- Exportul rezultatelor – am notat mai bine OMNeT++ pentru posibilitatea de prelucrarea a rezultatelor în *plove*. FACE nu dispune decât de o facilitate de editare de tip histogramă.
- Interfață pentru colectarea alarmelor – am notat mai bine FACE deoarece poate funcționa atât *off-line* cât și *on-line*.
- Timp de execuție – din informațiile publice (cf. *Figurii 11* din articolul [48]), asupra unui jurnal de 2900 de alarme de 36 de tipuri diferite, timpul de execuție al FACE a fost de 2 minute. Pentru un jurnal de alarme similar, OMNeT++, chiar fără procedurile de calcul a constrângerilor și de asamblare a rețelei Petri, a consumat un timp dublu. De aceea am notat mai bine proiectul FACE.

CONTRIBUŢII ŞI CONCLUZII

Pe parcursul elaborării tezei de doctorat am adus câteva contribuţii originale, atât teoretice cât şi practice, referitoare la problema analizei jurnalelor de alarme de telecomunicaţii. În acest capitol voi face o sinteză pentru a evidenţia contribuţiile originale care au fost prezentate în capitolele anterioare.

În încheiere, voi prezenta concluziile finale referitoare la acoperirea aspectelor principale ale problemei studiate, prin comentarea rezultatelor experimentale obţinute în raport cu rezultatele publicate la încheierea unui program de cercetare similar. Rezultatele experimentare obţinute reprezintă un suport de validare pentru soluţiile tehnice alese şi pentru contribuţiile teoretice exprimate în teză.

Contribuţii teoretice

Din punct de vedere teoretic, contribuţiile majore pe care le-am adus în problema analizei alarmelor în reţelele de telecomunicaţii sunt următoarele :

- Am îmbunătăţit algoritmului de recunoaştere a cronicilor de alarme prin extensia acestuia cu o procedură de calcul a constrângerilor temporale, introducând noţiuni despre extragerea de reguli de asociere.
- Am continuat exploatarea rezultatelor algoritmului prin asamblarea unei reţele Petri ale cărei tranziţii sunt etichetate de către cronicile frecvente recunoscute anterior de către algoritm. Asamblarea reţelei Petri se face în paralel cu procesul de recunoaştere a cronicilor frecvente şi permite astfel o analiză mai rapidă a cronicilor candidate pentru a găsi soluţii ale algoritmului.

În primul rând, pentru extensia algoritmului de recunoaştere a cronicilor frecvente, am reformulat algoritmul general de extragere de cronici de alarme frecvente, introdus în lucrarea [131]. Scopul principal al reformulării algoritmului a fost de a facilita scrierea acestuia în pseudo-limbaj de programare pentru ca, ulterior, să pot implementa acest algoritm într-un limbaj de programare de nivel înalt, cum este limbajul C++ în cadrul mediului de dezvoltare de aplicaţii informatice OMNeT++. Algoritmul este descris în paragraful §2.5 de la Capitolul 2.

Odată cu reformularea algoritmului am realizat şi descrierea unei proceduri de calcul pentru introducerea unor constrângeri temporale între alarme. Procedura de calcul, prezentată în paragraful §2.6.1, introduce noi variabile în aplicarea algoritmului, sub forma unor coeficienţi de încredere asociaţi regulilor de asociere, prezentaţi în §5.4. Prin introducerea coeficienţilor pentru exprimarea constrângerilor temporale între alarme, am

realizat o extensie a algoritmului de recunoaştere a cronicilor frecvente în jurnalele de alarme de telecomunicaţii.

Al doilea aspect major, din punct de vedere teoretic, este asamblarea unei reţele Petri ale cărei tranziţii să fie etichetate de cronicile frecvente de alarme determinate anterior prin aplicarea algoritmului. Scopul principal al asamblării unei astfel de reţele Petri este de a analiza noi aspecte ale cronicilor de alarme, în raport cu proprietăţile reţelelor Petri, prezentate în §4.4. Prin asamblarea reţelei Petri, în paralel cu recunoaşterea de cronici frecvente, se poate ajunge în anumite poziţii care exprimă interacţiuni de concurenţă sau de consecutivitate a două sau mai multe cronici. În cazul unor interacţiuni de concurenţă se vor putea exprima anumite dependinţe mai complexe între cronicile recunoscute, iar în cazul interacţiunilor de consecutivitate se determină faptul că acele cronici sunt relaţionate cauzal şi ordonate între ele. Analizând reţeaua Petri obţinută, se pot elimina din calcul anumite cronici frecvente care conduc la situaţii concurente sau consecutive. Se obţine astfel o îmbunătăţire a timpului de calcul al procesului de recunoaştere a cronicilor frecvente, anumite cronici candidate putând fi eliminate relativ devreme din soluţiile algoritmului.

Creşterea complexităţii soluţiei prin adăugarea modulului de asamblare a reţelei Petri este contrabalansată de o scădere a complexităţii algoritmului datorită eliminării unor soluţii care, deşi asamblate serial sau paralel de către algoritm, nu se verifică în reţeaua Petri. Rezultatele practice confirmă îmbunătăţirea timpului de găsire a soluţiei atunci când se adaugă execuţia în paralel a modulului de asamblare a reţelei Petri, pentru frecvenţe minime de apariţie deasupra unui anumit prag, după cum voi arăta în paragraful următor.

Contribuţii practice

Pentru a susţine cu argumente experimentale contribuţiile teoretice prezentate în paragraful precedent, am dezvoltat un proiect informatic care să realizeze procesul de analiză a alarmelor. Modulele informatice, dezvoltate după regulile de management ale unui proiect informatic, sunt descrise în Capitolul 6.

În raport cu contribuţia teoretică referitoare la extensia algoritmului de recunoaştere, am dezvoltat procedurile de calcul necesare implementării practice a algoritmului în mediul de programare OMNeT++. După cum am mai arătat, am ales mediul de programare OMNeT++ pentru a putea integra propriile mele module informatice, dezvoltate în limbajul de programare C++, care sunt prezentate în Anexa C.

Cele trei module informatice principale pe care le-am realizat sunt următoarele :

- Un modul pentru colectarea alarmelor şi construirea unui jurnal de alarme, bazat pe principiul unei achiziţii în timp real a alarmelor respectând anumite mecanisme uzuale de dispecerizare a fluxului de alarme.

- Un modul pentru prelucrarea alarmelor, care realizează algoritmul de recunoaştere a cronicilor frecvente.
- Un modul de asamblare a unor reţele Petri, pe măsură ce sunt recunoscute cronicile frecvente, în scopul verificării ipotezelor de lucru pentru cronicile candidate.

Alarmele şi cronicile de alarme sunt vehiculate între aceste module sub forma unor jetoane sau a unor schimburi de mesaje prin porţile de intrare şi ieşire ale modulelor informatice. Astfel, modulul de colectare gestionează o coadă de aşteptare de alarme, iar algoritmul va consuma evenimente din această coadă de aşteptare, urmând ca toate cronicile frecvente recunoscute să fie asamblate într-o reţea Petri. Experimentele realizate au arătat că algoritmul de recunoaştere, completat de asamblarea reţelei Petri, oferă un sistem puternic de analiză a jurnalelor de alarme de telecomunicaţii. Complexitatea algoritmului creşte odată cu scăderea frecvenţei minime de apariţie, deoarece atunci creşte numărul de cronici candidate care trebuie calculate.

Prin comparaţie cu implementarea similară rezultată în urma programului de cercetare *MAGDA (Modelisation et Apprentissage pour une Gestion Distribuée des Alarmes)* [136], se observă că proiectul meu informatic având implementat modulul Petri reuşeşte să îndeplinească funcţiile principale de recunoaştere a cronicilor şi de corelare a alarmelor, chiar dacă timpul de calcul consumat este mai mare decât în cazul implementării în mediul integrat FACE *(Frequency Analyzer for Chronicle Extraction)* [44], care este implementat pe o platformă mai puternică.

Concluzii finale

Concluziile prezentate la sfârşitul fiecărui capitol al tezei pot fi sintetizate pentru a oferi un cadru general asupra problemei studiate.

La sfârşitul Capitolului 1, unde am prezentat arhitectura unui sistem general de supervizare şi metodele de diagnosticare folosite în cazul reţelelor de telecomunicaţii, am arătat importanţa procesului de analiză a alarmelor. Datorită volumului mare de alarme, sistemul de supervizare ar putea fi solicitat să gestioneze un flux de alarme prea mare pentru a putea lua decizii de intervenţie în timp real. De aceea, este necesară prelucrarea alarmelor în sensul determinării unor secvenţe de alarme care se repetă în mod frecvent şi care se constituie astfel în cronici de alarme.

După cum am arătat în Capitolul 2, se pune problema recunoaşterii cronicilor frecvente în jurnalele de alarme. În urma unei analize care poate fi exprimată sub forma unui algoritm, se pot determina cronicile frecvente pe baza unor proceduri de generare a unor cronici candidate asupra cărora se efectuează un calcul al frecvenţei de apariţie. În acest algoritm am introdus o nouă procedură de calcul pentru a exprima constrângerile

temporale dintre alarme. Aplicând algoritmul vom obține o mulțime de cronici de alarme care apar cu o frecvență mai mare decât o valoare minimă considerată.

Modelarea matematică a jurnalelor de alarme, pentru aplicarea algoritmului prezentat în Capitolul 2, foloseşte teoria temporală numită algebra momentelor, introdusă în 1982 de McDermott [83]. În Capitolul 3 am prezentat o sinteză privind toate teoriile temporale sub aspectul analizei intervalelor şi a punctelor, pentru a evidenția motivele pentru care am ales algebra momentelor în cadrul analizei alarmelor din rețelele de telecomunicații. În finalul Capitolului 3 am ajuns la concluzia că modelul temporal cel mai potrivit pentru modelarea unor alarme de telecomunicații este modelul momentelor, deoarece acesta defineşte o colecție densă de puncte pe o linie temporală liniară la stânga şi ramificată în viitor. Colecția densă de puncte reprezintă alarmele, iar ramificarea în viitor reprezintă cronicile candidate. Am ales algebra momentelor deoarece conține cele două axiome importante de liniaritate şi de densitate, care exprimă cel mai bine alarmele în cazul rețelelor de telecomunicații.

Pentru a dezvolta analiza cronicilor de alarme, în Capitolul 4 am introdus noțiuni despre rețelele Petri. Aceste rețele permit o reprezentare simplă şi intuitivă a fenomenelor de concurență şi de cauzalitate. De aceea, considerând că tranzițiile rețelelor Petri sunt cronici de alarme, putem analiza relațiile de concurență sau de cauzalitate între aceste cronici. Concluzia principală din finalul Capitolului 4 este că rețelele Petri sunt foarte bine adaptate pentru analiza unor cronici de alarme. Cronicile de alarme pot fi definite atât pentru alarmele din rețelele de telecomunicații [32], [48], [131], cât şi pentru alarmele din sistemele de distribuție de energie electrică [72], [73].

Pentru a explica constrângerile temporale introduse în algoritmul de recunoaştere din Capitolul 2, în Capitolul 5 am prezentat cadrul teoretic al extragerii de reguli de asociere. Pornind de la analiza regulilor de asociere se pot construi reprezentări cu diagrame Hasse pentru a exprima cronicile frecvente. Diagramele Hasse sunt folosite pentru vizualizarea unor mulțimi parțial ordonate. Am folosit diagramele Hasse pentru a oferi o reprezentare grafică mai intuitivă asupra constrângerilor temporale calculate pentru cronicile candidate.

Aşa cum am mai arătat, am dezvoltat un sistem informatic pentru a implementa soluțiile tehnice şi a verifica dacă rezultatele experimentale confirmă o îmbunătățire a găsirii soluțiilor la aplicarea algoritmului. Experimentele realizate prin implementarea algoritmului într-un mediu de simulare a generării unor alarme au arătat că algoritmul este aplicabil cu succes asupra unor jurnale de alarme de o complexitate medie. Rezultatele sunt remarcabile pentru frecvențe minime de apariție corespunzător alese.

Explicația influenței alegerii frecvenței minime de apariție este următoarea:

Dacă frecvența minimă de apariție considerată este prea mare în raport cu numărul total de alarme, atunci algoritmul se va opri rapid, deja de la cronicile candidate de ordine de mărime mici. De exemplu, dacă avem un jurnal de alarme de 1.000 de

apariţii de alarme dintr-o familie de 100 de tipuri de alarme, alegerea unei frecvenţe minime de apariţie de 1.000 nu ar da nici un rezultat, algoritmul oprindu-se la primul pas, la calcularea frecvenţelor celor 100 de alarme.

Dacă frecvenţa minimă de apariţie este prea mică, atunci se poate ca algoritmul să determine foarte multe cronici candidate, ceea ce va încărca atât algoritmul cât şi modulul Petri, timpul de calcul pentru determinarea unei soluţii va creşte exponenţial, la fel ca şi complexitatea calculelor. De exemplu, pentru acelaşi jurnal de 1.000 de apariţii de alarme de 100 de tipuri diferite, dacă alegem frecvenţa minimă de apariţie egală cu 2, obţinem calcule extrem de complexe deoarece probabil toate cele 100 de alarme apar de cel puţin două ori şi mai departe se creează cronici candidate de ordinul 2 prin asamblarea paralelă sau serială a 100 de alarme, obţinând un volum foarte mare de cronici candidate. În plus, modulul Petri care funcţionează în paralel în scopul eliminării anumitor cronici frecvente, va fi la rândul lui supraîncărcat datorită numărului mare de cronici frecvente.

Pentru a avea un suport pentru contribuţiile originale aduse algoritmului, am construit trei strategii de execuţie a programului informatic:

- Executarea algoritmului de recunoaştere a cronicilor frecvente în forma iniţială, fără a apela procedura de calcul a constrângerilor temporale şi fără a apela modulul paralel de asamblare a reţelei Petri.
- Executarea algoritmului îmbunătăţit prin adăugarea procedurii de calcul a constrângerilor temporale
- Executarea completă a algoritmului şi a modulului de asamblare a reţelei Petri

Comparaţia între aceste 3 strategii de execuţie evidenţiază optimizarea timpului de determinare a soluţiei optime pe măsură ce se utilizează cele două proceduri menţionate. Rezultatele sunt comentate la sfârşitul Capitolului 6.

Astfel, folosind procedura de calcul a constrângerilor temporale, algoritmul elimină anumite cronici candidate înainte ca acestea să intre în procedura de calcul a frecvenţelor de apariţie. Acest lucru duce la o îmbunătăţire cu 5-10% a timpului de determinare a soluţiilor, în funcţie de frecvenţa minimă considerată.

Introducând în paralel modulul de asamblare a reţelei Petri, pe măsură ce se obţin de la algoritm soluţii sub forma cronicilor frecvente, modulul Petri va elimina anumite cronici care, deşi au fost determinate ca fiind frecvente, conduc la situaţii de consecutivitate sau de concurenţă în reţeaua Petri. Deoarece se elimină cronici frecvente, soluţiile parţiale nu mai sunt aceleaşi, dar se poate observa îmbunătăţirea timpului de determinare a soluţiilor finale cu până la 5%, fără pierderi de informaţii utile. Eliminarea unora dintre soluţiile parţiale ale algoritmului, deşi acestea au fost determinate ca fiind frecvente, se datorează faptului că aceste soluţii parţiale sunt incluse în soluţii parţiale de ordin superior, deci în cronici frecvente de ordin superior. Soluţia sau soluţiile finale obţinute sunt aceleaşi, cu sau fără aplicarea modulului Petri, doar dacă ele nu sunt

redundante. Modulul Petri oferă astfel soluţii calitativ superioare pentru problema recunoaşterii cronicilor frecvente.

O ultimă remarcă importantă este aceea că unele dintre cronicile recunoscute dintr-un jurnal real de alarme au putut fi explicate prin influenţa unor factori care nu ţin de domeniul telecomunicaţiilor. De exemplu, într-o reţea de telecomunicaţii analizată existau anumite disfuncţiuni în calitatea serviciilor oferite de reţea, datorită creşterii temperaturii de operare a echipamentelor din reţea. De fapt, am descoperit o cronică de alarme referitoare la instalaţia de răcire a aerului în raport cu întreruperea unei surse secundare de alimentare a unui echipament din reţea. Instalaţia de răcire a aerului era conectată la sursa secundară de alimentare care a fost întreruptă. Ambele alarme, atât cea privind întreruperea alimentării electrice cât şi cea privind temperatura ridicată a sistemului, apăreau în jurnalul de alarme dar nu erau în domeniul de analiză al expertului care superviza respectiva reţea de telecomunicaţii. În general, astfel de alarme, etichetate uneori drept alarme *colaterale*, sunt rareori modelizate de către experţii în supervizarea reţelelor de telecomunicaţii, deoarece nu depind în mod direct de domeniul lor de activitate.

Tema de cercetare poate fi continuată în direcţia evoluţiei algoritmului pentru a putea analiza duratele alarmelor. După cum am mai precizat, duratele alarmelor pot fi descrise în raport cu momentele de apariţie şi de dispariţie ale unei alarme în reţea. În acest sens, se poate utiliza un model matematic construit pe o teorie temporală bazată pe intervale de timp, nu numai pe puncte sau momente.

O altă posibilă continuare a temei de cercetare constă în dezvoltarea unui sistem predictiv de cronici de alarme. Sistemul predictiv de cronici de alarme ar trebui să identifice prezenţa unei cronici analizând parţial alarmele recepţionate. Dacă se poate determina, după primele alarme recepţionate, că va urma o secvenţă de alarme care a fost deja recunoscută, atunci sistemul predictiv ar putea cere modulului de colectare sau modulului de analiză a alarmelor să ignore următoarele alarme. Astfel, s-ar putea reduce fluxul de alarme între modulele de colectare şi analiză, eliminându-se cel puţin o parte din fluxul care corespunde unor cronici deja recunoscute. De exemplu, să presupunem că o anumită cronică, formată dintr-o succesiune de 10 alarme, a fost deja recunoscută în sistem ca fiind o cronică frecventă. Reprezentarea fizică a acestei cronici de alarme ar putea fi întreruperea unui circuit de comunicaţie, fapt care conduce la întreruperile apelurilor telefonice care se efectuau pe acel circuit, urmate de alarme ale maşinilor logice care pilotau acele apeluri telefonice. Atunci când se detectează alarma corespunzătoare întreruperii circuitului, care este la originea cronicii de alarme, modulul care realizează sistemul predictiv poate transmite modulelor de colectare sau de analiză a alarmelor că va urma o succesiune de alarme, care a fost recunoscută anterior ca fiind o cronică frecventă. Sistemul predictiv de cronici de alarme poate constitui un prim pas în direcţia dezvoltării unui sistem expert care să utilizeze cronicile de alarme recunoscute.

Experimentele realizate au arătat că algoritmul de recunoaştere a cronicilor, completat de procedura de calcul a constrângerilor temporale şi de procedura de analiză a unei reţele Petri etichetată de cronicile frecvente anterior descoperite, este capabil să gestioneze jurnale de alarme de dimensiuni reale.

Rezultatele experimentale, obţinute în urma unor analize asupra unor jurnale de alarme reale, au fost prezentate unor experţi în domeniul supervizării reţelelor de telecomunicaţii. Concluzia acestor experţi a fost că sistemul dezvoltat este cu siguranţă un instrument de mare ajutor în procesul de supervizare al reţelelor de telecomunicaţii şi ar putea sta la baza dezvoltării unui sistem expert pentru supervizarea acestor reţele.

ANEXA A : ACRONIME

- **ACE - Adaptive Communication Environment** : Interfață de comunicare în C++, în regim open-source, dezvoltată de Douglas C. Schmidt şi un grup de cercetători de la University of California Irvine. Există şi o versiune Java denumită JACE.
- **API - Application Programming Interface** : Interfață prin care un program informatic accesează sistemul de operare sau anumite servicii ale acestuia. API-ul este definit la nivelul codului sursă al nucleului sistemului de operare şi asigură portabilitatea aplicațiilor informatice pe mai multe platforme de dezvoltare.
- **CCITT - Comitetul Consultativ Internațional pentru Telefonie şi Telegrafie** : Abreviere pentru organizația internațională al cărei obiect de activitate este definirea de standarde în domeniul comunicațiilor de date. Începând din data de 1.03.1993 organizația devine ITU-T.
- **CHILL - CCITT High Level Language** : Limbaj de programare în timp real dedicat dezvoltării sistemelor de telecomunicații, standardizat de CCITT. CHILL a fost dezvoltat în anii 1970 şi evoluat până în anul 1996.
- **CNET - Centre National d'Etudes des Télécommunications** : Centrul național francez pentru cercetări în domeniul telecomunicațiilor. CNET are sediul la Lannion, Franța.
- **CRS - Chronicle Recognition System** : Sistem informatic de extragere a unor secvențe de evenimente, dezvoltat la CNET.
- **FACE - Frequency Analyzer for Chronicle Extraction** : Sistem informatic dezvoltat în urma proiectului MAGDA între Alcatel, Francetelecom şi INRIA, pentru analiza frecvenței alarmelor de telecomunicații.
- **FTP – File Transfer Protocol** : Standard de comunicație pentru transferul de fişiere informatice de la diferite sisteme de operare. FTP este un protocol client-server, definit de standardul RFC 959, publicat în1985.
- **GUI - Graphical User Interface** : Interfață grafică cu utilizatorul care facilitează interacțiunea prin elemente grafice, spre deosebire de *command line interface* unde interacțiunea cu utilizatorul se face prin schimburi de linii de caractere.
- **IFA - Institut de Formation Alcatel** : Organism de pregătire profesională, în regim de educație continuă, în cadrul companiei Alcatel.
- **INRIA - Institut National de Recherche en Informatique et en Automatique** : Institut francez pentru cercetări în domeniul ştiinței calculatoarelor, automaticii şi matematicii aplicate. Dezvoltă mai multe arii de cercetare, având un centru de cercetare la Rennes, denumit IRISA, Institut National de Recherche en Informatique et Automatique.

- **ITU - International Telecommunication Union** : Organism de standardizare pentru telefonie şi date, care realizează sesiuni plenare, odată la 4 ani, pentru adoptarea de noi standarde în acest domeniu. ITU are sediul la Geneva, Elveţia.
- **LFN - Logical File Name** : Fişier informatic pentru colectarea anumitor rezultate şi prezentarea acestora către operator sau către un sistem de supervizare.
- **MAGDA - Modelisation et Apprentissage pour une Gestion Distribuée des Alarmes** : Proiect comun Alcatel, Francetelecom şi IRISA pentru cercetarea modelării alarmelor în reţelele de telecomunicaţii.
- **MAPI - Messaging Application Programming Interface** : Bibliotecă de funcţii dezvoltată pentru a permite sistemelor de operare să acceseze sisteme de mesagerie electronică.
- **MONET - Mobile Networks and Applications** : Proiect informatic pentru integrarea serviciilor din reţelele de telecomunicaţii mobile.
- **NED - Network Description Language** : Limbaj de descriere a topologiei reţelelor din mediul integrat de dezvoltare OMNeT++, pentru facilitarea implementării interfeţelor de comunicaţie ale modulelor informatice.
- **NMC - Network Management Centre** : Centru de supervizare a reţelelor de telecomunicaţii. Punct central de la care poate fi controlată reţeaua de telecomunicaţii, constituit în general din mai multe centre de operare şi mentenanţă.
- **OMNeT++ - Objective Modular Network Testbed in C++** : Mediu integrat de dezvoltare de proiecte informatice, iniţiat la Universitatea Tehnică din Budapesta şi evoluat în regim de surse libere. OMNeT++ este disponibil în mediul academic fără a necesita licenţe informatice.
- **ODP - Open Distributed Processing** : Arhitectură standardizată a nivelului aplicaţie definit de ISO. ODP este o evoluţie naturală pentru standardizarea construcţiei de sisteme distribuite.
- **OCB - Organe de Commutation Binaire** : Abreviere pentru denumirea iniţială dată centralelor telefonice Alcatel 1000 E10 dezvoltate la Lannion, Franţa.
- **RHM - Relations Homme-Machine** : Interfaţă cu utilizatorul folosind comenzi simple alfanumerice, structurate pe mai multe familii funcţionale, bazate pe dicţionare de mnemonici.
- **TMM - Time Map Manager** : Algebră temporală propusă de Dean şi McDermott în 1987 pentru a defini o reţea de puncte temporale şi relaţiile metrice dintre aceste momente.
- **SDL - Specification Design Language** : Limbaj de specificaţie şi descriere a comportamentului sistemelor de telecomunicaţii, standardizat de ITU prin recomandarea Z.100, folosind extensii ale unor automate cu stări finite, FSM (Finite State Machines)

ANEXA B : TOPOLOGIA REȚELEI OMNET++

Versiunea utilizată : OMNeT++ 3.1 source (tgz) (versiunea 3.1 făcută publică la data de 31.03.2005)

Ultima versiune testată : OMNeT++ 3.2 win32 binary (exe) (versiunea 3.2pre4 publică la 20.10.2005)

Topologia rețelei OMNeT++ folosită pentru verificarea proiectului informatic este cea din *Figura Anexa B.1* :

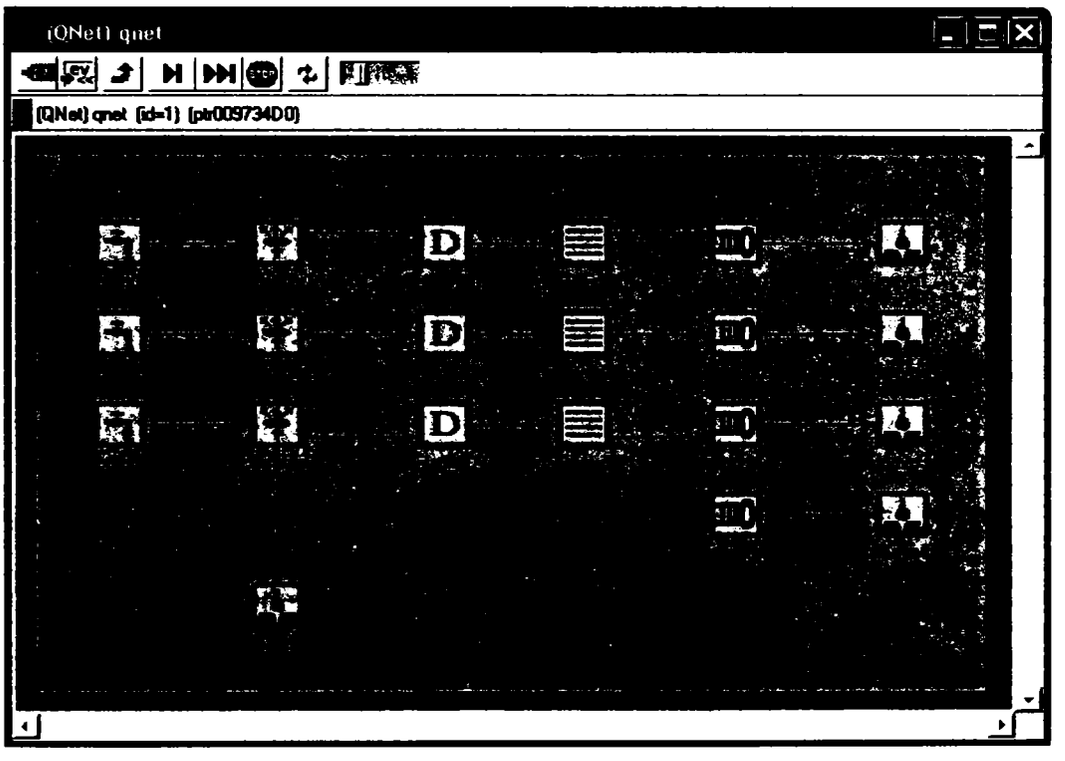


Figura Anexa B.1 Topologia rețelei OMNeT++

Modulele *classifier* implementează dispecerul de colectare de alarme și modulele *qserver* realizează funcțiile de analiză a alarmelor (algoritmul de recunoaștere a cronicilor frecvente, procedura de introducere a constrângerilor temporale, procedura de asamblare și analiză a rețelei Petri având tranzițiile etichetate de cronicile frecvente recunoscute).

Modulele *leave* implementează funcțiile de vizualizare a rezultatelor experimentale obținute. Modulele *delay* și *passiveq* realizează colectarea alarmelor prin modul de transfer *pull*. Modulele *enter* generează populația de evenimente (alarme) direct de pe platforma rețelei de telecomunicații, înainte de a fi înregistrate de modulele colectoare de alarme.

Pentru a realiza diferitele scenarii de testare am inhibat sau am activat diferite module simple implementate în modulele *qserver* din această topologie.

ANEXA C : ACE (ADAPTIVE COMMUNICATION ENVIRONMENT)

ACE [135], [139], [140] este o dezvoltare în limbajul de programare C++ al unei biblioteci de elemente fundamentale pentru funcții de comunicare, având suport pentru o largă varietate de sisteme de operare (AIX, HP-UX, Linux, Solaris, Win32, VxWorks etc.).

ACE conține peste 135.000 de linii de cod în limbajul de programare C++, realizate în urma unui efort colectiv, timp de mai mulți ani, al unor experți în domeniul sistemelor de operare și al serviciilor oferite la acest nivel. ACE este în continuare portat pe diferite noi sisteme de operare și pe noi platforme de dezvoltare.

Biblioteca ACE oferă următoarele funcții de comunicare :

- Demultiplexarea evenimentelor și dispecerizarea fluxului de evenimente
- Inițializarea serviciilor
- Comunicarea între procese și gestionarea memoriei partajate
- Rutarea mesajelor
- Reconfigurarea dinamică a serviciilor distribuite și multi-threading

Componentele elementare ale lui ACE și relațiile ierarhice sunt reprezentate în diagrama din *Figura Anexa C.1*:

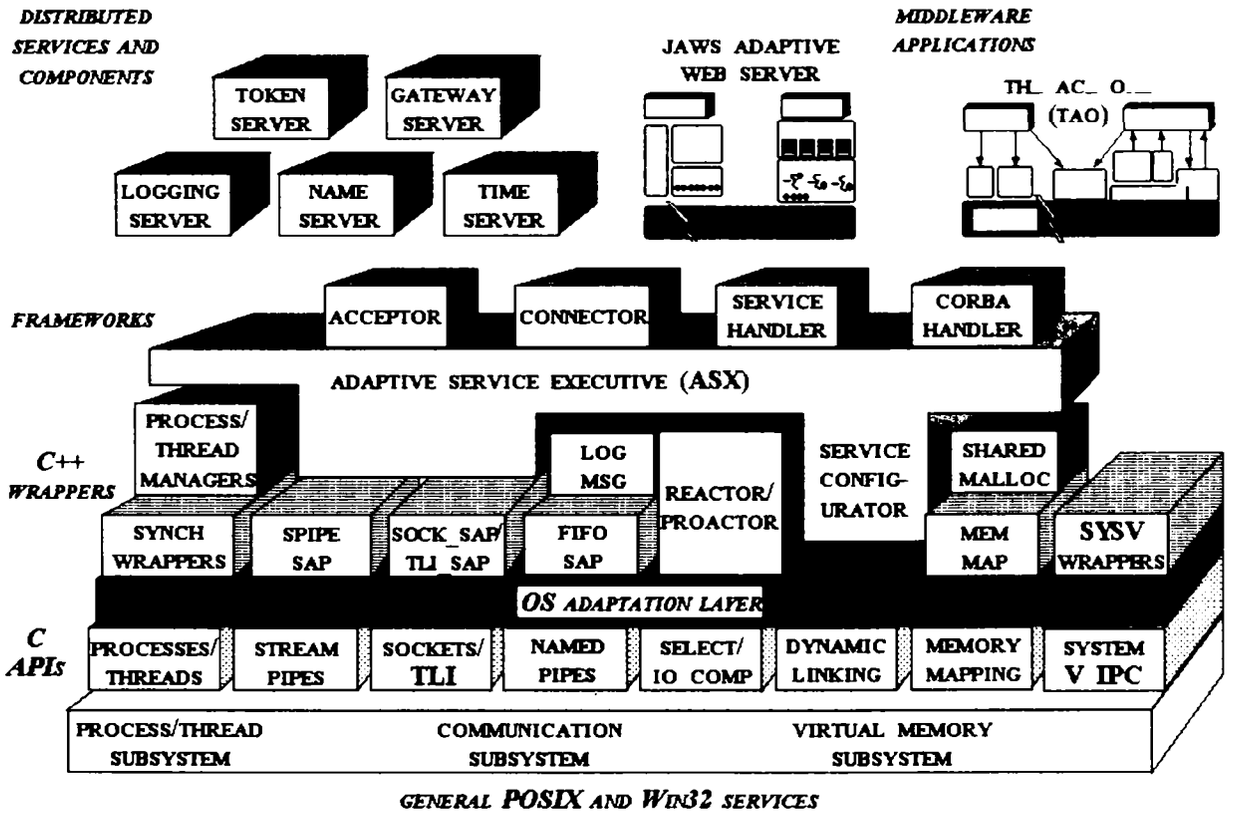


Figura Anexa C.1 Diagrama componentelor elementare ACE

În această anexă sunt listate sursele informatice implementate în limbajul de programare C++ din cadrul bibliotecii ACE. Pentru a nu încărca *Anexa B* cu listing-ul ACE în C++ am ales să prezint clasele ACE prin legături web (hyperlink) către acest listing, disponibil la adresa web http://www.huihoo.org/ace_tao/ :

[CORBA]

- CORBA_Ref.cpp
- CORBA_Ref.h
- CORBA_Ref.i

[Containers]

- Array.cpp
- Array.h
- Array.i
- Containers.cpp
- Containers.i
- Containers.h
- Hash_Map_Manager.cpp
- Hash_Map_Manager.h
- Filecache.cpp
- Filecache.h
- Free_List.cpp
- Free_List.i
- Free_List.h
- Managed_Object.cpp
- Managed_Object.h
- Managed_Object.i
- Map_Manager.cpp
- Map_Manager.h
- Map_Manager.i
- Object_Manager.cpp
- Object_Manager.i
- Object_Manager.h
- SString.cpp
- SString.h
- SString.i

[Concurrency]

- Activation_Queue.h
- Activation_Queue.cpp
- Atomic_Op.i
- Future.h
- Future.cpp
- Method_Request.h
- Method_Request.cpp
- Process.cpp
- Process.h
- Process.i
- Process_Manager.cpp
- Process_Manager.h
- Process_Manager.i
- Sched_Params.cpp
- Sched_Params.h
- Sched_Params.i
- Synch.cpp
- Synch.h
- Synch.i
- Synch_Options.cpp
- Synch_Options.h
- Synch_Options.i
- Synch_T.cpp
- Synch_T.h
- Synch_T.i
- Thread.cpp
- Thread.h
- Thread.i
- Thread_Manager.cpp
- Thread_Manager.h
- Thread_Manager.i
- Token.cpp
- Token.h
- Token.i

[Config]

- config.h
- Basic_Types.cpp
- Basic_Types.h
- Basic_Types.i
- Version.h

[Connection]

- Acceptor.cpp
- Acceptor.h
- Acceptor.i
- Asynch_Acceptor.cpp
- Asynch_Acceptor.h
- Asynch_Acceptor.i
- Asynch_IO.cpp
- Asynch_IO.h
- Asynch_IO.i
- Connector.cpp
- Connector.h
- Connector.i
- Dynamic_Service.cpp
- Dynamic_Service.h
- Dynamic_Service.i
- Strategies.cpp
- Strategies.h
- Strategies.i
- Strategies_T.cpp
- Strategies_T.h
- Strategies_T.i
- Svc_Handler.cpp
- Svc_Handler.h
- Svc_Handler.i

[IPC]

- IO_SAP.cpp
- IO_SAP.h
- IO_SAP.i
  DEV.cpp
- DEV.h
- DEV.i
- DEV_Connector.cpp
- DEV_Connector.h
- DEV_Connector.i
- DEV_IO.cpp
- DEV_IO.h
- DEV_IO.i
- TTY_IO.cpp
- TTY_IO.h
- FILE.cpp
- FILE.h
- FILE.i
- FILE_Connector.cpp
- FILE_Connector.h
- FILE_Connector.i
- FILE_IO.cpp
- FILE_IO.h
- FILE_IO.i
- IPC_SAP.cpp
- IPC_SAP.h
- IPC_SAP.i
- Addr.cpp
- Addr.h
- Addr.i
- DEV_Addr.cpp
- DEV_Addr.h
- DEV_Addr.i
- FILE_Addr.cpp
- FILE_Addr.h
- FILE_Addr.i
- INET_Addr.cpp
- INET_Addr.h
- INET_Addr.i
- SPIPE_Addr.cpp
- SPIPE_Addr.h
- SPIPE_Addr.i
- UNIX_Addr.cpp
- UNIX_Addr.h
- UNIX_Addr.i
- UPIPE_Addr.h

[FIFO_SAP]

- FIFO.cpp
- FIFO.h
- FIFO.i
- FIFO_Recv.cpp
- FIFO_Recv.h
- FIFO_Recv.i
- FIFO_Recv_Msg.cpp
- FIFO_Recv_Msg.h
- FIFO_Recv_Msg.i
- FIFO_Send.cpp
- FIFO_Send.h

- FIFO_Send.i
- FIFO_Send_Msg.cpp
- FIFO_Send_Msg.h
- FIFO_Send_Msg.i

[SOCK_SAP]

- LOCK_SOCK_Acceptor.cpp
- LOCK_SOCK_Acceptor.h
- LSOCK.cpp
- LSOCK.h
- LSOCK.i
- LSOCK_Acceptor.cpp
- LSOCK_Acceptor.h
- LSOCK_Acceptor.i
- LSOCK_CODgram.cpp
- LSOCK_CODgram.h
- LSOCK_CODgram.i
- LSOCK_Connector.cpp
- LSOCK_Connector.h
- LSOCK_Connector.i
- LSOCK_Dgram.cpp
- LSOCK_Dgram.h
- LSOCK_Dgram.i
- LSOCK_Stream.cpp
- LSOCK_Stream.h
- LSOCK_Stream.i
- SOCK.cpp
- SOCK.h
- SOCK.i
- SOCK_Acceptor.cpp
- SOCK_Acceptor.h
- SOCK_Acceptor.i
- SOCK_CODgram.cpp
- SOCK_CODgram.h
- SOCK_CODgram.i
- SOCK_Connector.cpp
- SOCK_Connector.h
- SOCK_Connector.i
- SOCK_Dgram.cpp
- SOCK_Dgram.h
- SOCK_Dgram.i
- SOCK_Dgram_Bcast.cpp
- SOCK_Dgram_Bcast.h
- SOCK_Dgram_Bcast.i
- SOCK_Dgram_Mcast.cpp
- SOCK_Dgram_Mcast.h
- SOCK_Dgram_Mcast.i
- SOCK_IO.cpp
- SOCK_IO.h
- SOCK_IO.i
- SOCK_Stream.cpp
- SOCK_Stream.h
- SOCK_Stream.i

[SPIPE_SAP]

- SPIPE.cpp
- SPIPE.h
- SPIPE.i
- SPIPE_Acceptor.cpp
- SPIPE_Acceptor.h
- SPIPE_Acceptor.i
- SPIPE_Connector.cpp
- SPIPE_Connector.h
- SPIPE_Connector.i
- SPIPE_Stream.cpp
- SPIPE_Stream.h
- SPIPE_Stream.i

[TLI_SAP]

- TLI.cpp
- TLI.h
- TLI.i
- TLI_Acceptor.cpp
- TLI_Acceptor.h
- TLI_Acceptor.i
- TLI_Connector.cpp
- TLI_Connector.h
- TLI_Connector.i
- TLI_Stream.cpp
- TLI_Stream.h
- TLI_Stream.i

[UPIPE_SAP]

- UPIPE_Acceptor.cpp

- UPIPE_Acceptor.h
- UPIPE_Acceptor.i
- UPIPE_Connector.cpp
- UPIPE_Connector.h
- UPIPE_Connector.i
- UPIPE_Stream.cpp
- UPIPE_Stream.h
- UPIPE_Stream.i

[Misc]

- IOStream.cpp
- IOStream.h
- IOStream_T.i
- Pipe.cpp
- Pipe.h
- Pipe.i
- Signal.cpp
- Signal.h
- Signal.i

[Logging and Tracing]

- Dump.cpp
- Dump.h
- Dump_T.cpp
- Dump_T.h
- Log_Msg.cpp
- Log_Msg.h
- Log_Msg.i
- Log_Priority.h
- Log_Record.cpp
- Log_Record.h
- Log_Record.i
- Trace.cpp
- Trace.h
- Trace.i

[Mem_Map]

- Mem_Map.cpp
- Mem_Map.h
- Mem_Map.i

[Shared_Malloc]

- Malloc.cpp
- Malloc.h
- Malloc.i
- Malloc_T.cpp
- Malloc_T.h
- Malloc_T.i
- Memory_Pool.cpp
- Memory_Pool.h
- Memory_Pool.i

[Shared_Memory]

- Shared_Memory.h
- Shared_Memory_MM.cpp
- Shared_Memory_MM.h
- Shared_Memory_MM.i
- Shared_Memory_SV.cpp
- Shared_Memory_SV.h
- Shared_Memory_SV.i

[Utils]

- Obstack.cpp
- Obstack.h
- Read_Buffer.cpp
- Read_Buffer.h
- Read_Buffer.i

[Misc]

- ARGV.cpp
- ARGV.h
- ARGV.i
- Auto_Ptr.cpp
- Auto_Ptr.h
- Auto_Ptr.i
- Date_Time.cpp
- Date_Time.h
- Date_Time.i
- Dynamic.cpp
- Dynamic.h
- Dynamic.i

- Get_Opt.cpp
- Get_Opt.h
- Get_Opt.i
- Registry.cpp
- Registry.h
- Singleton.cpp
- Singleton.h
- Singleton.i
- System_Time.cpp
- System_Time.h

[Name_Service]

- Local_Name_Space.cpp
- Local_Name_Space.h
- Local_Name_Space_T.cpp
- Local_Name_Space_T.h
- Name_Proxy.cpp
- Name_Proxy.h
- Name_Request_Reply.cpp
- Name_Request_Reply.h
- Name_Space.cpp
- Name_Space.h
- Naming_Context.cpp
- Naming_Context.h
- Registry_Name_Space.cpp
- Registry_Name_Space.h
- Remote_Name_Space.cpp
- Remote_Name_Space.h

[OS Adapters]

- ACE.cpp
- ACE.h
- ACE.i
- OS.cpp
- OS.h
- OS.i

[Reactor]

- Event_Handler.cpp
- Event_Handler.h
- Event_Handler.i
- Event_Handler_T.cpp
- Event_Handler_T.h
- Event_Handler_T.i
- Handle_Set.cpp
- Handle_Set.h
- Handle_Set.i
- Priority_Reactor.cpp
- Priority_Reactor.i
- Priority_Reactor.h
- Proactor.h
- Proactor.i
- Proactor.cpp
- Reactor.cpp
- Reactor.h
- Reactor.i
- Reactor_Impl.h
- Select_Reactor.cpp
- Select_Reactor.h
- Select_Reactor.i
- WFMO_Reactor.cpp
- WFMO_Reactor.h
- WFMO_Reactor.i
- XtReactor.cpp
- XtReactor.h

[Service_Configurator]

- DLL.cpp
- DLL.h
- Parse_Node.cpp
- Parse_Node.h
- Parse_Node.i
- Service_Config.cpp
- Service_Config.h
- Service_Config.i
- Service_Manager.cpp
- Service_Manager.h
- Service_Manager.i
- Service_Object.cpp
- Service_Object.h
- Service_Object.i
- Service_Repository.cpp
- Service_Repository.h
- Service_Repository.i

- Service_Types.cpp
- Service_Types.i
- Service_Types.h
- Shared_Object.cpp
- Shared_Object.h
- Shared_Object.i
- Svc_Conf.h
- Svc_Conf_l.cpp
- Svc_Conf_y.cpp
- Svc_Conf_Tokens.h

[Streams]

- IO_Cntl_Msg.cpp
- IO_Cntl_Msg.h
- IO_Cntl_Msg.i
- Message_Block.cpp
- Message_Block.h
- Message_Block.i
- Message_Queue.cpp
- Message_Queue.h
- Message_Queue.i
- Message_Queue_T.cpp
- Message_Queue_T.h
- Message_Queue_T.i
- Module.cpp
- Module.h
- Module.i
- Multiplexor.cpp
- Multiplexor.h
- Multiplexor.i
- Stream.cpp
- Stream.h
- Stream.i
- Stream_Modules.cpp
- Stream_Modules.h
- Stream_Modules.i
- Task.cpp
- Task.h
- Task.i
- Task_T.cpp
- Task_T.h
- Task_T.i

[System_V_IPC]

- SV_Message.cpp
- SV_Message.h
- SV_Message.i
- SV_Message_Queue.cpp
- SV_Message_Queue.h
- SV_Message_Queue.i
- Typed_SV_Message.cpp
- Typed_SV_Message.h
- Typed_SV_Message.i
- Typed_SV_Mess_Queue.cpp
- Typed_SV_Mess_Queue.h
- Typed_SV_Mess_Queue.i

[System_V_Semaphores]

- SV_Semaphore_C.cpp
- SV_Semaphore_Complex.h
- SV_Semaphore_Complex.i
- SV_Semaphore_S.cpp
- SV_Semaphore_Simple.h
- SV_Semaphore_Simple.i

[System_V_Shared_Memory]

- SV_Shared_Memory.cpp
- SV_Shared_Memory.h
- SV_Shared_Memory.i

[Timers]

- High_Res_Timer.cpp
- High_Res_Timer.h
- High_Res_Timer.i
- Profile_Timer.cpp
- Profile_Timer.h
- Profile_Timer.i
- Time_Request_Reply.cpp
- Time_Request_Reply.h
- Time_Value.h
- Timer_Hash.cpp
- Timer_Hash.h
- Timer_Hash_T.cpp

- Timer_Hash_T.h
- Timer_Heap.cpp
- Timer_Heap.h
- Timer_Heap_T.cpp
- Timer_Heap_T.h
- Timer_List.cpp
- Timer_List.h
- Timer_List_T.cpp
- Timer_List_T.h
- Timer_Queue.cpp
- Timer_Queue.h
- Timer_Queue.i
- Timer_Queue_Adapters.cpp
- Timer_Queue_Adapters.h
- Timer_Queue_Adapters.i
- Timer_Queue_T.cpp
- Timer_Queue_T.h
- Timer_Queue_T.i
- Timer_Wheel.cpp
- Timer_Wheel.h
- Timer_Wheel_T.cpp
- Timer_Wheel_T.h

[Token_Service]

- Local_Tokens.cpp
- Local_Tokens.h
- Local_Tokens.i
- Remote_Tokens.cpp
- Remote_Tokens.h
- Remote_Tokens.i
- Token_Collection.cpp
- Token_Collection.h
- Token_Collection.i
- Token_Manager.cpp
- Token_Manager.h
- Token_Manager.i
- Token_Request_Reply.cpp
- Token_Request_Reply.h
- Token_Request_Reply.i
- Token_Invariants.h
- Token_Invariants.i
- Token_Invariants.cpp

ANEXA D : LISTING-UL PROIECTULUI INFORMATIC (MODULELE PRINCIPALE)

Alarm_analyzer.cpp

```
//-------------------------------------------------------------------------------
// AUTHOR  : Serafin P
// PROJECT : OMNeT++ Alarm Analyzer
// DOMAIN  : Program Principal
//-------------------------------------------------------------------------------
@file Alarm_analyzer.cpp
@brief program principal
//-------------------------------------------------------------------------------
#ifndef __ALARM_ANALYZER_cc__
#define __ALARM_ANALYZER_cc__

#include <sys/resource.h>
#include "ALARM_ANALYZERMessage.h"
#include "ace/Service_Config.h"
#include "ace/Reactor.h"
#include "ALARM_ANALYZER_Configurator.h"
#include "InternalErrno.h"
#include "platformsupapi.h"
#include "ALARM_ANALYZERCOMMON.h"
#include "OmaTrace.h"
#include "Signal_Event_Handler.h"

int ACE_TMAIN(int argc, ACE_TCHAR *argv[])
{
        int status,local;
        // autorizare coredump
        struct rlimit rlp ;
            getrlimit ( RLIMIT_CORE, &rlp ) ;
            if (rlp.rlim_cur == 0 )
                {
                rlp.rlim_cur = rlp.rlim_max ;
                setrlimit ( RLIMIT_CORE, &rlp ) ;
                }
        // transfer optiuni API (ptr debug si trase)
        int res = platformSupGetOpt(argc,argv);
        // lectura fisier de configuratie
        ALARM_ANALYZER_Configurator::Init();
        // init parametrare OmaLog : aplicatie + LOG
        OmaLog::setappliName("Alarm_Analyzer");
        status = ALARM_ANALYZER_Configurator::GetLong ("ALARM_ANALYZER_LOG ", local);
        if ((status == ServerConfigurator::UNKNOWN_SYMBOL) ||
            (status == ServerConfigurator::CONVERT_ERROR))
           {
           local = LOG_LOCAL0;
           }
        OmaLog::setlocal(local);
        ALARM_ANALYZER_LOG_NOTICE(1,"log_service " << NB_MAX)
    // start aplicatie debug (daca e parametrata)
    #ifndef ALARM_ANALYZER_NTRACE
        string traceFile;
    if (!ALARM_ANALYZER_Configurator::GetString("A      ANALYZER_TRACE    ",traceFile))
        {
       OmaSard::ServerTrace::CreateTheServerTrace(traceFile.c_str(),0); // no flip/flop
       OmaSard::ServerTrace::GetTheServerTrace().enable()
       int omaTraceLevel;
    // initializarea nivel trase
    string omaTraceLevelStr;
    int retcode = ALARM_ANALYZER_Configurator::GetString("TRACE_LEVEL",omaTraceLevelStr);
    if (!retcode)
       {
       omaTraceLevel = strtol(omaTraceLevelStr.c_str(),NULL,NB_MAX);
       }
        else
       {
```

```
        omaTraceLevel = Trace::FILTER_NONE;
       }
       OmaSard::ServerTrace::GetTheServerTrace().setFilter(omaTraceLevel);
   }
#endif

//-------------------------------------------------------------------------------
// initializare servicii ACE
//-------------------------------------------------------------------------------
// apelare configurator de servicii (ptr evitare conflicte)
   char *aceArgv[3];
   aceArgv[0] = argv[0];
   aceArgv[1] = "-f";              // flag modificare fisier de configurare
   aceArgv[2] = ". Alarm_Analyzer_Conf/Alarm_Analyzer.conf";
        // fisier de configurare ACE
   int nberror = ACE_Service_Config::open (3,aceArgv);
   if (nberror)
   {
       MAIN_TRACE("eroare servicii ACE");
       ALARM_ANALYZER_LOG_ERROR(ERRNO_ALARM_ANALYZER_CONFIG_ERROR,
        "Services loading error");
       exit(E_PLATFORMSUP_BADCONFIGURATION);
   }
   MAIN_TRACE("main() return from ACE_Service_Config");

//-------------------------------------------------------------------------------
// event loop (gestiune servicii ACE)
//-------------------------------------------------------------------------------
// abonament la semnale de evenimente SIGUSR1
        Signal_Event_Handler sigHandler;
        ACE_Sig_Handler handler;
        handler.register_handler(SIGUSR1, &sigHandler);
        //   ACE_Reactor::instance()->run_reactor_event_loop();
   while (1)
   {
       MAIN_TRACE("main() before : ");
       int retval = ACE_Reactor::instance()->handle_events();
       MAIN_TRACE("main() after : " << retval);
    }
   return 0;
}
#endif
```

Chronicle_ManagementImpl.cpp

```
//-------------------------------------------------------------------------------
// AUTHOR  : Serafin P
// PROJECT : OMNeT++ Alarm Analyzer
// DOMAIN  : Net Management
//-------------------------------------------------------------------------------
@file ChronicleManagementImpl.cpp
@brief interfata algoritmului
//-------------------------------------------------------------------------------
#ifndef __ChronicleManagementImpl_cc__
#define __ChronicleManagementImpl_cc__

#include <stdio.h>
#include "ChronicleManagementImpl.hh"
#include "ALARM_ANALYZER_Configurator.hh"
#include "AlarmClasses.hh"
#include "ALARM_ANALYZER_Service_Object.hh"
#include "ChronicleService.hh"
#include "PETRI_Service_Task.hh"
#include "OmaIdlException.hh"
#include "OMNETErrorCodeC.h"
#include "OMNETFrameIdC.h"
#include "Frame.hh"
#include "CommonError.hh"
#include "TemporaryDifferedHandle.hh"
#include "CMD.hh"
```

```
#include "ActlErrorCode3.h"
#include "CHRONICLEClient.hh"
#include "CHRONICLEClientFactory.hh"
#include "PETRI_Automaton.hh"
#include "ActlImpl.hh"
#include "ChronicleFrame.hh"
#include <unistd.h>

string ChronicleManagementImpl::_LnkNameChronicleSessionCpt = "";
unsigned long ChronicleManagementImpl::ChronicleSessionCpt =
ChronicleManagementImpl::initChronicleSessionCpt_i();
int ChronicleManagementImpl::firstSessionInd = 1;
int ChronicleManagementImpl::_fistCHRONICLECommand =0;

//-------------------------------------------------------------------------
// constructor
//-------------------------------------------------------------------------
ChronicleManagementImpl::ChronicleManagementImpl()
{
   int l_ret = 0;
   // test prima sesiune, return imediat daca indicatorul neactualizat in destructor
   if (firstSessionInd)
   {
       PETRI_TRACE("ChronicleManagementImpl ... eManagementImpl ");
       return;
   }
   try
   {
       // recuperarea serviciului CHRONICLE
       ALARM_ANALYZER_Service_Object *ChronicleService;
       ChronicleService = ALARM_ANALYZER_Service_Object::GetTheServiceObject
(PETRI_SERVICE_NAME);
       Int_ProtSession = NULL;
       client = NULL;
       firstAlarm = true;
       if (ChronicleService)
       {
       PETRI_Service_Task *ChronicleServiceTask;
       ChronicleServiceTask = (PETRI_Service_Task*)ChronicleService->getServiceTask();
       if (ChronicleServiceTask)
           {
             // deschidere sesiune
             ChronicleSessionFactory = ChronicleServiceTask->getSessionFactory();
             Int_ProtSession = new PETRI_INT_PROTsession((ActlImpl::InterfaceImpl*)this);
             int retopenINT_PROTSync;
             if (ChronicleSessionFactory)
     retopenINT_PROTSync = ChronicleSessionFactory->openINT_PROTSync(Int_ProtSession);
               // creare  client asociat automatului
             if (retopenINT_PROTSync == 0) {
     ChronicleClientFactory = ChronicleServiceTask->getClientFactory();
     l_ret = ChronicleClientFactory -> addCHRONICLEClient(&client);
             } else {
              ChronicleSessionFactory->removeSession(Int_ProtSession);
                   }
              // crearea automatului si initializarea clientului
             if( (!retopenINT_PROTSync) && (l_ret == 0))
             {
              ChronicleSessionId = getNewChronicleSessionCptVal_i();
              automaton = new PETRI_Automaton;
       client -> setSessionId(Int_ProtSession);
       client -> setAutomatonId(automaton);
       Int_ProtSession -> setCHRONICLECLient(client);
              return;
               }
               else
               {
       OmaSard::OmaIdlException exceptionINT_PROTko (OMNET::UNABLE_TO_OPEN_THE_SESSION);
       exceptionINT_PROTko.addFrame(OMNET::OMNET_UNABLE_TO_OPEN_THE_SESSION_FRAME_ID);
       OmaSard::StringFrameAttribute serviceName(PETRI_SERVICE_NAME);
       exceptionINT_PROTko.addAttributeToLastFrame(serviceName);
```

```
        PETRI_TRACE "ChronicleManagement::...
: "  << exceptionINT_PROTko.getStringDescription());
          throw exceptionINT_PROTko.getPlatformError();
                }
            }
        }
        OmaSard::OmaIdlException exception(OMNET::SERVICE_NOT_STARTED);
        exception.addFrame(OMNET::OMNET_SERVICE_NOT_STARTED_FRAME_ID);
        OmaSard::StringFrameAttribute serviceName(PETRI_SERVICE_NAME);
        exception.addAttributeToLastFrame(serviceName);
        PETRI_TRACE("ChronicleManagementImpl::...
: " << exception.getStringDescription());
        throw exception.getPlatformError();
    }
    catch(Platform::Error error)
    {
        PETRI_TRACE("Platform::Error: " << error);
        throw error;
    }
    catch (System::Exception)
    {
        PETRI_TRACE("SystemException ");
        throw OmaSard::ComSystemError();
    }
}  // deschidere sesiune de comunicare

//-------------------------------------------------------------------------------
// destructor
//-------------------------------------------------------------------------------
ChronicleManagementImpl::~ChronicleManagementImpl()
{
    // test prima sesiune si init control
    if (firstSessionInd)
    {
         firstSessionInd = 0;
         PETRI_TRACE("ChronicleManagementImpl::ChronicleManagementImpl          ");
         return;
    }
     // destructor automat
     delete automaton;
     // stergere ChronicleSessionId din topologie
     TemporaryDifferedHandle::deleteChronicleSessionID(ChronicleSessionId);
     PETRI_TRACE("ChronicleManagementImpl::~ChronicleManagementImpl
TemporaryDifferedHandle\n" << TemporaryDifferedHandle::dump());
     // destructor client
     ChronicleClientFactory -> removeCHRONICLEClient(client);
     // inchidere sesiune
     if (Int_ProtSession -> getState() == INT_PROTsession::activ)
     ChronicleSessionFactory -> closeINT_PROTSync(Int_ProtSession);
     PETRI_TRACE("ChronicleManagementImpl::~ChronicleManagementImpl Int_ProtSession
"<<Int_ProtSession);
     ChronicleSessionFactory -> removeSession(Int_ProtSession);
}

//-------------------------------------------------------------------------------
// start ptr metode
//-------------------------------------------------------------------------------
void ChronicleManagementImpl::ChronicleCommand
        (const OMNET::Chronicle::Command_Argument & argument,
        OMNET::Chronicle::ChronicleCallBack_ptr callback,
        OMNET::Invoke_Id invokeId, const char * systemId)
     ACE_THROW_SPEC ((CORBA::SystemException, Platform::Error))
{
    try
    {
     TDIRFASU refAuto;
     const char *   data;
     unsigned long len;
     unsigned short classArray;
     string className = "";
     int nbAuthClass = 0;
```

```
    string mode="";          // control
    char com[7];
    size_t  taille;
    const char * delims = "::";
    PETRI_TRACE("                                    << this <<          ');
    if (_fistCHRONICLECommand == 0 )
        {  // citire o singura data la prima cerere
              PETRI_TRACE("                                                  ');
          CMD::ReadCMD();
        _fistCHRONICLECommand =1;
        }

    // recuperarea claselor din fisierul CMD
       PLATFORM_ACCESS_CONTROL_BEGIN_METHOD();
   if (firstAlarm)  // UserName and SessionName
   {
        // referinta ALARM_ANALYZER in mesaj sesiune catre client CORBA
  Int_ProtSession -> setRefAlarm_Analyzer(ActlImpl::InterfaceImpl::getUserName().data());
        // creare element in retea
   TemporaryDifferedHandle::createChronicleSessionID
  (ActlImpl::InterfaceImpl::getSessionName().data(), ChronicleSessionId);
        PETRI_TRACE("ChronicleManagementImpl::ChronicleCommand Temp                    "
                    << TemporaryDifferedHandle::dump());
        firstAlarm = false;
   }
   ALARM_ANALYZER_Configurator::GetString("          ",mode);
   if (mode == "ALARM")
   {
        // verificare autorizare
   } else
   {
        // recuperarea clasei modulului
        tail = strcspn(argument.info, delims);
        if (tail <= 6) {
            strncpy(com, argument.info, tail);
            com[tail] = '\0';
        } else {
        PETRI_TRACE("ChronicleManagementImpl::ChronicleManagementImpl                 ");
            Platform::ParamList params;
            throw Platform::Error(Platform::AccessControl::UNAUTHORIZED_METHOD, params);
// exception defined in ActlErrorCode.idl
        }
        PETRI_TRACE("ChronicleManagementImpl::ChronicleCommand com.xml = "<< com <<
dimensiune :" << tail);
        classArray = CMD::GetClass(string(com));
        for (int i = 0; i < 16; i++) {
            if ((classArray >> i) & 1) {
                char number[4];
                sprintf(number,"%d",i+1);
                string className = string(ALARM_CLASSE_PREFIX) + string(number);
                PETRI_TRACE("Cerere autorizare: " << className << "    " << com);
                // daca autorizat increment nbAuthClass
                try
                {
                    PLATFORM_ACCESS_CONTROL_BEGIN_METHOD();
                    ActlImpl::InterfaceImpl::beginMethod(className);
                    ActlImpl::InterfaceImpl::endMethod(className);
                    nbAuthClass++;
                    PETRI_TRACE(className << " nbAuthClass = " << nbAuthClass);
                }
                catch(Platform::Error error)
                {
                    PETRI_TRACE(error << " " << className << "  ");
                }
            }
        }
    }
    // daca nici o clasa nu este autorizata "unauthorized method"
    if (nbAuthClass == 0) {
        PETRI_TRACE("ChronicleManagementImpl::ChronicleManagementImpl : Neautorizat : " <<
PETRI_SERVICE_NAME);
```

```
        Platform::ParamList params;
        throw Platform::Error(Platform::AccessControl::UNAUTHORIZED_METHOD, params);
exception
    }
    // verificarea automatului
    if (automaton -> ChronicleCommand() == -1) {
        PETRI_TRACE "
" << PETRI_SERVICE_NAME);
        OmaSard::OmaIdlException exception(OMNET::UNEXPECTED_METHODE);
        exception.addFrame(OMNET::OMNET_UNEXPECTED_METHODE_FRAME_ID);
        OmaSard::StringFrameAttribute serviceName(PETRI_SERVICE_NAME);
        exception.addAttributeToLastFrame(serviceName);
        throw exception.getPlatformError();
    }
    // test IOR receptionat
    OMNET::Chronicle::ChronicleCallBack_ptr test =
OMNET::Chronicle::ChronicleCallBack::_narrow(callback);
    if (CORBA::is_nil(test))
    {
        CORBA::release(test);
        // IOR receptionat eronat LOG
        ALARM_ANALYZER_LOG_WARNING (ERRNO_PETRI_BAD_IOR, "                                    " );
        // emisie exceptie
        OmaSard::SymbolicFrameAttribute paramId("
        OmaSard::StringFrameAttribute value("");
        OmaSard::ComBadParameterValue except(paramId, value);
        PETRI_TRACE("emisie exceptie : " << except.getStringDescription());
        throw OmaIdlException::GetPlatformError(except);
    }
    CORBA::release(test);
    // adresare raspuns
    client -> setIOR(callback);
    client -> setInvokeId(invokeId);
    client -> setSystemId(systemId);
    // format mesaj sesiune
    memset(&refAuto, '\0', sizeof(TDIRFASU));
    strncpy(refAuto.sdedif, argument.accessSystem,
              OMNET::Chronicle::ACCESS_SYSTEM_NAME_MAX_SIZE);
    switch(argument.activator._d())
    {
        case OMNET::AT_OPERATOR :
            strncpy(refAuto.nopInt_Prot, (const char *)
                argument.activator.operatorName(), OMNET::OPERATOR_NAME_MAX_SIZE);
            break;
        case OMNET::AT_NMU_APPLICATION :
            refAuto.nopInt_Prot[0] = RefAuto_NMC;
            strncpy(&refAuto.nopInt_Prot[1], (const char *)
                argument.activator.nmuAppli().nmuAppliName, OMNET::MMU_APPLI_NAME_SIZE);
            for (int i = 0; i< OMNET::MMU_APPLI_INFO_SIZE; i++)
            {
                 refAuto.nopInt_Prot[OMNET::MMU_APPLI_NAME_SIZE + i + 1] =
argument.activator.nmuAppli().nmuAppliInfo[i];
            }
            break;
        case OMNET::AT_OTHER_APPLICATION :
            refAuto.nopInt_Prot[0] = RefAuto_OTHER;
            strncpy(&refAuto.nopInt_Prot[1], (const char
*)argument.activator.otherAppli().otherAppliName, OMNET::OTHER_APPLI_NAME_SIZE);
            for (int i = 0; i< OMNET::OTHER_APPLI_INFO_SIZE; i++)
            {
                 refAuto.nopInt_Prot[OMNET::OTHER_APPLI_NAME_SIZE + i + 1] =
argument.activator.otherAppli().otherAppliInfo[i];
            }
            break;
        default :    // exceptie
            break;
    }
    strncpy(refAuto.refcde, argument.command, OMNET::COMMAND_NAME_MAX_SIZE);
    // Chronicle id
    memcpy(refAuto.nevjo, &ChronicleSessionId, sizeof(ChronicleSessionId));
    if (ChronicleSessionFactory -> getSocketState() == INT_PROTsessionFactory::ES)
```

```
    {
        len = strlen(argument.info);
        data = argument.info;
        Int_ProtSession -> PETRI_request(&refAuto, (char *)data, len);
    } else
    {
        OmaSard::OmaIdlException exception(OMNET::UNABLE_TO_OPEN_THE_SESSION);
        exception.addFrame(OMNET::OMNET_UNABLE_TO_OPEN_THE_SESSION_FRAME_ID);
        OmaSard::StringFrameAttribute serviceName(PETRI_SERVICE_NAME);
        exception.addAttributeToLastFrame(serviceName);
        PETRI_TRACE("ChronicleManagementImpl::Chronicle
<< exception.getStringDescription());
        throw exception.getPlatformError();
    }
   }
   catch(Platform::Error error)
   {
       PETRI_TRACE("Platform::Error " << error);
       throw error;
   }
    PETRI_TRACE("ChronicleManagementImpl::Chronicle          " << this <<          ");
}

//-------------------------------------------------------------------------
// raspundere la o cerere (validare sau relansare)
//-------------------------------------------------------------------------
void ChronicleManagementImpl::ChronicleExtraInfoReply (const
OMNET::Chronicle::Extra_Info_Result & result, OMNET::Invoke_Id invokeId,
             const char * systemId)
     ACE_THROW_SPEC ((CORBA::SystemException, Platform::Error))
{
   const char * data;
   unsigned long len;
   PETRI_TRACE("ChronicleManagementImpl::Chro icleE      foReply " << result.type);
   try
   {
        if (ChronicleSessionFactory -> getSocketState() == INT_PROTsessionFactory::ES)
        {
            len = strlen(result.informations);
            data = result.informations;
            PETRI_TRACE("stare automat "<< automaton -> dump());
            if ((result.type == OMNET::QT_CONFIRM)
                &&
                ((automaton -> ChronicleValidationReply()) == 1)) {
                Int_ProtSession -> PETRI_validation_reply((char *)data, len);
                return;
            } else
            if ((result.type == OMNET::QT_EXTRAINFO)
                &&
                ((automaton -> ChronicleExtraInfoReply()) == 1)) {
                Int_ProtSession -> PETRI_additional_line_reply((char *)data, len);
                return;
            } else {
               OmaSard::OmaIdlException exception(OMNET::UNEXPECTED_METHODE);
                exception.addFrame(OMNET::OMNET_UNEXPECTED_METHODE_FRAME_ID);
                OmaSard::StringFrameAttribute serviceName(PETRI_SERVICE_NAME);
                exception.addAttributeToLastFrame(serviceName);
                PETRI_TRACE("ChronicleManagementImpl::ChronicleManagementIdl
/easteptat : " << exception.getStringDescription());
                throw exception.getPlatformError();
            }
        } else {
            OmaSard::OmaIdlException exception(OMNET::UNABLE_TO_OPEN_THE_SESSION);
            exception.addFrame(OMNET::OMNET_UNABLE_TO_OPEN_THE_SESSION_FRAME_ID);
            OmaSard::StringFrameAttribute serviceName(PETRI_SERVICE_NAME);
            exception.addAttributeToLastFrame(serviceName);
            PETRI_TRACE("ChronicleManagementImpl::ChronicleManagementIdl    INT_PROT %
" << exception.getStringDescription());
            throw exception.getPlatformError();
        }
   }
```

```
   catch(Platform::Error error)
   {
       PETRI_TRACE("SystemException ");
       throw OmaSard::ComSystemError();
   }
}

//-------------------------------------------------------------------------------
// abandonare candidate
//-------------------------------------------------------------------------------
void ChronicleManagementImpl::ChronicleCancel (OMNET::Invoke_Id invokeId, const char *
systemId)
   ACE_THROW_SPEC ((CORBA::SystemException, Platform::Error))
{
   PETRI_TRACE("ChronicleManagementImpl::ChronicleCancel ");
   try
   {
        if (ChronicleSessionFactory -> getSocketState() == INT_PROTsessionFactory::ES)
        {
         if ((automaton -> ChronicleCancel()) == -1) {      // verif state machine
          OmaSard::OmaIdlException exception(OMNET::UNEXPECTED_METHODE);
         exception.addFrame(OMNET::OMNET_UNEXPECTED_METHODE_FRAME_ID);
         OmaSard::StringFrameAttribute serviceName(PETRI_SERVICE_NAME);
         exception.addAttributeToLastFrame(serviceName);
         PETRI_TRACE("ChronicleManagementImpl::ChronicleCancel
: " << exception.getStringDescription());
         throw exception.getPlatformError();
            } else
                Int_ProtSession -> PETRI_abort();
        } else
        {
            OmaSard::OmaIdlException exception(OMNET::UNABLE_TO_OPEN_THE_SESSION);
            exception.addFrame(OMNET::OMNET_UNABLE_TO_OPEN_THE_SESSION_FRAME_ID);
            OmaSard::StringFrameAttribute serviceName(PETRI_SERVICE_NAME);
            exception.addAttributeToLastFrame(serviceName);
            PETRI_TRACE("ChronicleManagementImpl::ChronicleCancel
" << exception.getStringDescription());
            throw exception.getPlatformError();
        }
   }
   catch(Platform::Error error)
   {
       PETRI_TRACE("SystemException ");
       throw OmaSard::ComSystemError();
   }
}

//-------------------------------------------------------------------------------
// sfarsit metode
//-------------------------------------------------------------------------------
//-------------------------------------------------------------------------------
// implementare interfata
//-------------------------------------------------------------------------------
ActlImpl::InterfaceImpl* ChronicleManagementImpl::CreateImplInstance()
{
    ActlImpl::InterfaceImpl* interfaceI;
    interfaceI = new ChronicleManagementImpl();
    PETRI_TRACE("ChronicleManagementImpl::CreateImplInstance " << interfaceI);
    return interfaceI;
}
string*  ChronicleManagementImpl::GetMethodes()
{
   string *methodes;
   int methodesNumber =0;
   string mode="";
   ALARM_ANALYZER_Configurator::GetString("ACTL_MODE",mode);
   if (mode == "ALARM")
   {
       // declararea tuturor comenzilor
       methodesNumber = CMD::GetALARMNumber();
       methodes = CMD::GetALARMList();
```

```
        for (int j = 0; j < methodesNumber; j++)
        PETRI_TRACE("[illegible] " << j << " " << methodes[j]);
    }
    else
    {
        // declararea tuturor claselor
        methodesNumber = MAX_ALARM_CLASSE_NUMBER-MIN_ALARM_CLASSE_NUMBER+1;
        methodes = new string[methodesNumber];
        for (int i = MIN_ALARM_CLASSE_NUMBER; i<= MAX_ALARM_CLASSE_NUMBER; i++)
        {
            char number[4];
            sprintf(number,"%d",i);

            string claseName = string(ALARM_CLASSE_PREFIX) + string(number);
            methodes[i-MIN_ALARM_CLASSE_NUMBER] = claseName;
            PETRI_TRACE("PETRI_Service_Impl::[illegible]" << i << " = " << claseName);
        }
    }
    return methodes;
}
int ChronicleManagementImpl::GetMethodesNumber()
{
    int methodesNumber =0;
    string mode="";
    ALARM_ANALYZER_Configurator::GetString("[illegible]",mode);
    if (mode == "ALARM")
   {
        methodesNumber = CMD::GetALARMNumber();
    }
    else
    {
         methodesNumber = MAX_ALARM_CLASSE_NUMBER-MIN_ALARM_CLASSE_NUMBER+1;
    }
     return methodesNumber;
}

//-----------------------------------------------------------------------------
// static method,  initChronicleSessionCpt_i()
//-----------------------------------------------------------------------------
unsigned long  ChronicleManagementImpl::initChronicleSessionCpt_i()
{
        unsigned long ret =  1; // valoare initiala pt secvente
        char buffer[10000];
        int nb_read;
        PETRI_TRACE(" [illegible]");
        if (ALARM_ANALYZER_Configurator::GetString(SERVICE_PETRI_SESSIONCPT_FILE_NAME_LNK,
ChronicleManagementImpl::_LnkNameChronicleSessionCpt) != ServerSConfigurator::SUCCEED)
                {
        ChronicleManagementImpl::_LnkNameChronicleSessionCpt =
DefaultChronicleSessionCpt_LNK_NAME;
        PETRI_TRACE(" [illegible]
_LnkNameChronicleSessionCpt nu este in configurator!! " <<
DefaultChronicleSessionCpt_LNK_NAME);
        ALARM_ANALYZER_LOG_ERROR (ERRNO_ALARM_ANALYZER_SESSIONCPT_FILE_CFG, \
        " [illegible]
" << DefaultChronicleSessionCpt_LNK_NAME);
                } // nu este in configurator
        nb_read = readlink (ChronicleManagementImpl::_LnkNameChronicleSessionCpt.data(),
buffer, sizeof(buffer) -1 );
        if (nb_read == -1)
                {
                PETRI_TRACE(" [illegible]
nb_read = -1 ");
                }
        else if ( nb_read >= sizeof(buffer) -1)
                {
                PETRI_TRACE(" [illegible]
long string in return from system call ");
                }
        else
                {
```

```
                buffer[nb_read] = '\0';
                sscanf(buffer, "%ld", &ret);
                }
        PETRI_TRACE(" ChronicleMa                                                    << ret );
        return ret;
}
//-------------------------------------------------------------------------------
//      getNewChronicleSessionCptVal_i
//-------------------------------------------------------------------------------
unsigned long  ChronicleManagementImpl::getNewChronicleSessionCptVal_i()
{
        char buffer[1000];
        static ACE_Thread_Mutex ChronicleSessionCptMutex(0,0);
        int ret =0;
        PETRI_TRACE(" ChronicleManagementImpl::getN                                  ");
        ACE_GUARD_RETURN (ACE_Thread_Mutex, local_guard, ChronicleSessionCptMutex , 0);
        // get a new value
        ChronicleManagementImpl::ChronicleSessionCpt++;
        if (ChronicleManagementImpl::ChronicleSessionCpt == 0) // 0 = reserved
                ChronicleManagementImpl::ChronicleSessionCpt++;
        sprintf(buffer, "%ld", ChronicleManagementImpl::ChronicleSessionCpt);
        // create a name equal to current id in file system
        ret = unlink(ChronicleManagementImpl::_LnkNameChronicleSessionCpt.data());
        if (ret == -1)
                {
                PETRI_TRACE(" ChronicleManagementImpl::getNew
ret -1 from OMNET call, buffer is " << buffer  \
                << " _LnkNameChronicleSessionCpt is " <<
ChronicleManagementImpl::_LnkNameChronicleSessionCpt);
                }
        ret = symlink (buffer ,
ChronicleManagementImpl::_LnkNameChronicleSessionCpt.data() );
        if (ret == -1)
                {
                PETRI_TRACE(" ChronicleManagementImpl::getNewChronicleSessionC
ret -1 from OMNET call, buffer is " << buffer  \
                << " _LnkNameChronicleSessionCpt is " <<
ChronicleManagementImpl::_LnkNameChronicleSessionCpt);
                }
        else
                {
                PETRI_TRACE(" ChronicleManagementImpl::getNewChronicleSes
ret OK from OMNET call, buffer is " << buffer  \
                << " _LnkNameChronicleSessionCpt is " <<
ChronicleManagementImpl::_LnkNameChronicleSessionCpt);
                }
        PETRI_TRACE(" ChronicleManagementImpl::getNewChronicleSessionCptVal          ret
val " << ChronicleManagementImpl::ChronicleSessionCpt);
return ChronicleManagementImpl::ChronicleSessionCpt;
}
#endif
```

CMD.cpp

```
//-------------------------------------------------------------------------------
// AUTHOR  : Serafin P
// PROJECT : OMNeT++ Alarm Analyzer
// DOMAIN  : Common File
//-------------------------------------------------------------------------------
@file CMD.cpp
@brief modul comun
//-------------------------------------------------------------------------------
#ifndef __CMD_cc__
#define __CMD_cc__

#include "CMD.hh"
#include "ebcdicconverter.hh"
#include "ALARM_ANALYZER_Configurator.hh"
#include "ALARM_ANALYZERCOMMON.hh"
#include "CHRONICLEErrno.hh"
#include <iostream>
```

```
#include <fstream>
using namespace std;

//-------------------------------------------------------------------------------
// dimensiune header
//-------------------------------------------------------------------------------
#define HEADER_SIZE    1

//-------------------------------------------------------------------------------
// structura unui articol
//-------------------------------------------------------------------------------
#pragma pack (push, 1)
typedef struct element {
#include "CMDCMD.H"
} element;
#pragma pack(pop)

map <string, unsigned short>  CMD::_AlarmClass;
unsigned long                 CMD::_AlarmNumber;
string *                      CMD::_AlarmList;
string                        CMD::_CMD;

//-------------------------------------------------------------------------------
// Scop : citirea unui fisier pentru construirea topologiei retelei
// Etape :
//      citirea PATH in fisierul de configurare
//      deschiderea fisierului de topologia ca fisier binar
//      ignorarea header-ului
//      while operatiile asupra topologiei sunt OK
//          citirea unui articol
//          setarea string-ului EBCDIC
//          transformarea EBCDIC - ASCII
//          asocierea unei clase pentru articolul ASCII
//      inchiderea fisierului
//      construirea retelei din lista de articole
//-------------------------------------------------------------------------------
void CMD::ReadCMD(void)
{
   string ebcdicStr = "";
   char   command[7];
   string asciiStr = "";

   element article;
   map <string, unsigned short>::iterator i;
   int  rg = 0;
   ifstream CMD_fic;
   int len;

   if (ALARM_ANALYZER_Configurator::GetString("CMD_PATH",_cmd) !=
ServerConfigurator::SUCCEED){
       _cmd = "cmd";
   }
   _AlarmNumber = 0;
   cmd_fic.open(_cmd.data(), ios::binary);
   cmd_fic.seekg(0, ios_base::end);
   len = cmd_fic.tellg();
   if ((len == 0) || (len == -1))
       ALARM_ANALYZER_LOG_WARNING(ERRNO_ALARM_ANALYZER_EMPTY_CMD, "
inexistent");
   CMD_fic.seekg(HEADER_SIZE, ios::beg);
   int indpr=0;
   int nbtotalko=0;
   while (cmd_fic.good())
    {
       cmd_fic.read((char *)&article, sizeof(article));
       indpr++;
#ifdef onconservecontrolePSTROLD
       if ((article.pstr != 0)
           && (article.pstr != (short)0x0071)
           && (article.pstr != (short)0xFF00)
            && (article.pstr != (short)0x000B))
```

```
        {
            PETRI_TRACE("CMD::ReadCMD article nr. "<< article.nord);
            ALARM_ANALYZER_LOG_DEBUG(ERRNO_ALARM_ANALYZER_BAD_CMD, "
<< 0x0000, 0xFF00, 0x0008, 0x0071");
        }
        else
#endif
        {
            strncpy(command, article.cmde, sizeof(article.cmde));
            for (int j = 5; j > 0; j--)
               {
                if (command[j] == (char)0x40)
                    command[j] = 0;
                else
                    break;
                }
            command[6] = (char)0; // EOF string

#ifdef onkeepcontrolPSTROLD
            if (       (article.pstr == (short)0)
                ||     (article.pstr == (short)0x0071)
                )
#endif
            if ( (article.pstr >> 8 ) == (short)0)
                {
                ebcdicStr = command;
                if (ebcdicStr != "")
                    {
                    asciiStr = "" + EBCDICConverter::toASCII(ebcdicStr);
                    _AlarmClass[asciiStr] = article.clas;
                    _AlarmNumber++;
                    } // string non vid
                else
                    {
                       // additionnal debug
                       PETRI_TRACE( "CMD::ReadCMD article nr " << indpr << ",
string vid : " \ << ALARM_ANALYZERCOMMON::decodeHexa( (char *)&article , sizeof(article)
, NULL, 0, 30, 4)) ;
                       nbtotalko++;
                    }
                } // pstr 0000 sau 0071
                else
                    {
                       // additionnal debug
                       PETRI_TRACE( "CMD::ReadCMD article nr " << indpr << ",          \
                       << ALARM_ANALYZERCOMMON::decodeHexa( (char *)&article ,
sizeof(article) , NULL, 0, 30, 4)) ;
                    PETRI_TRACE("CMD::ReadCMD article nr. " << indpr << ",
primul octet pstr : " << hex  << article.pstr << dec );     */
                    nbtotalko++;
                    }
         } // else pstr rejectat
   } // end while
   CMD_fic.close();
   PETRI_TRACE("CMD::Read CMD terminat, numar total de articole citit " << indpr << ",
dintre care "<< nbtotalko  << " articole ignorate "  );
   _AlarmList = new string[_AlarmNumber];
   for (i = _AlarmClass.begin(); i != _AlarmClass.end(); i++)
       _AlarmList[rg++] = i->first;
   PETRI_TRACE(dump());
}
//-----------------------------------------------------------------------------
// obtinerea string-ului ASCII referitor la cronici
// return 0 daca nu este in topologie
//-----------------------------------------------------------------------------
unsigned short CMD::GetClass(string command_)
{
    map <string, unsigned short >::iterator i;
    for (i = _AlarmClass.begin(); i != _AlarmClass.end(); i++) {
       if (i->first == command_) {
           return i->second;
```

```
        }
    }
    return (unsigned short)0;
}

//-------------------------------------------------------------------------------
// obtinerea numarului modulului din topologie
//-------------------------------------------------------------------------------
unsigned long CMD::GetALARMNumber(void)
{
    return _AlarmNumber;
}

//-------------------------------------------------------------------------------
// obtinerea listei de module
//-------------------------------------------------------------------------------
string * CMD::GetALARMList(void)
{
    return _AlarmList;
}

//-------------------------------------------------------------------------------
// dump pentru contextul topologiei
//-------------------------------------------------------------------------------
string CMD::dump(void)
{
    map<string, unsigned short >::iterator i;
    string str = "";
    char ligne[80];

    str = "\nCMD clase de comenzi din fisier"; str += "\n"; str += _cmd;
    if (_AlarmNumber == 0)
        str += "\nNici un articol in fisierul CMD";
    else
        for (i = _AlarmClass.begin(); i != _AlarmClass.end(); i++) {
            sprintf(ligne,"\n%s - %d", (i->first).data(), i->second);
            str += ligne;
        }
    return str;
}
#endif
```

CHRONICLEClientFactory.cpp

```
//-------------------------------------------------------------------------------
// AUTHOR  : Serafin P
// PROJECT : OMNeT++ Alarm Analyzer
// DOMAIN  : Generarea de cronici candidate
//-------------------------------------------------------------------------------
@file CHRONICLEClientFactory.cpp
@brief generare cronici candidate
//-------------------------------------------------------------------------------
#include "CHRONICLEClientFactory.hh"
#include "CHRONICLEClient.hh"
#include "ALARM_ANALYZERCOMMON.hh"

//-------------------------------------------------------------------------------
// constructor
//-------------------------------------------------------------------------------
CHRONICLEClientFactory::CHRONICLEClientFactory(ServiceINT_PROTId s)
: ALARM_ANALYZERClientFactory(s)
{
    identServ = s;
    PETRI_TRACE("CHRONICLEClientFactory::CHRONICLEClientFactory(ServiceINT_PROTId s)");
}

//-------------------------------------------------------------------------------
// destructor
//-------------------------------------------------------------------------------
CHRONICLEClientFactory::~CHRONICLEClientFactory()
{
```

```
    PETRI_TRACE("CHRONICLEClientFactory::                    ");
}

int  CHRONICLEClientFactory::addCHRONICLEClient (CHRONICLEClient
**ppALARM_ANALYZERClient_)
{
    CHRONICLEClient* pClient;
    int ret = 0;

    pClient = new CHRONICLEClient(identServ, this -> _ALARM_ANALYZERSenderFactory);

    PETRI_TRACE("CHRONICLEClient**ppALARM_ANALYZERClient_:                    " << pClient);

    if (pClient != NULL)
        {
        // adaugarea unui client in lista clasei mostenite
        ret = addALARM_ANALYZERClient (pClient);
        if (ret != 0)
                {
                delete  pClient;
                }
        }
     else
        {
        ret =  -1;
        }

     (*ppALARM_ANALYZERClient_) = pClient;
     return ret;

} // addCHRONICLEClient()

int CHRONICLEClientFactory::removeCHRONICLEClient(CHRONICLEClient* client)
{
    return removeALARM_ANALYZERClient( client);
}

string CHRONICLEClientFactory::dump(void)
{
    string str;
    str = "CHRONICLE client factory\n"; str += ALARM_ANALYZERClientFactory::dump();
    return str;
}
```

Colector.cpp

```
//-----------------------------------------------------------------------------
// AUTHOR   : Serafin P
// PROJECT : OMNeT++ Alarm Analyzer
// DOMAIN   : Colector de alarme
//-----------------------------------------------------------------------------
@file Colector.cpp
@brief modul collector de alarme

#include "IAS_Ala_Spon_Send.hh"
#include "ALARM_ANALYZERMacros.hh"
//-----------------------------------------------------------------------------
//      IAS_Ala_Spon_Send : constructor
//      Descriere : alocarea memoriei pentru sesiune
//-----------------------------------------------------------------------------
IAS_Ala_Spon_Send::IAS_Ala_Spon_Send()
{
    _message = new SASSDType;
    memset(_message, 0, sizeof(SASSDType));
    INT_PROTmessage::setHeaderAddress((char *)_message);

    return;
}
//-----------------------------------------------------------------------------
//     ~OpenRequest : destructor
//-----------------------------------------------------------------------------
```

```
IAS_Ala_Spon_Send::~IAS_Ala_Spon_Send()
{
    IAS_TRACE("IAS_Ala_Spon_Send::~IAS                                    ");
    delete _message;
    return;
}

//-------------------------------------------------------------------------
//  setCommonHeader : init header
//-------------------------------------------------------------------------
void IAS_Ala_Spon_Send::setCommonHeader (const char * idOr_, const char * idDe_,
                                const char * reOr_, const char * reDe_, long reSe_, char
* idSe_)
{
    IAS_TRACE("IAS_Ala_Spon_Send::set        Spon_Send :                          "
<<sizeof(SASSDType))

        INT_PROTmessage::setHeader(sizeof(SASSDType), CCHJSASSD, idOr_, idDe_, reOr_,
reDe_, reSe_, idSe_);

    return;
}
//-------------------------------------------------------------------------
//  setIAS_Ala_Spon_Send : set header si parametrii
//-------------------------------------------------------------------------
void IAS_Ala_Spon_Send::setIAS_Ala_Spon_Send(short nat, short cat, int nenr, int codef,
                        char * comment, char * info_, const char chaine)
{
     char buf[] = {' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
',' ',' '};
        IAS_TRACE("IAS_Ala_Spon_Send::setIAS_Ala_Spon_Send :                  ");
    _message->id=2;
    _message->natal=nat;
    _message->catal=cat;
    _message->tyal=20;
    _message->codef=codef;
    _message->nenr=nenr;
    _message->typfor=1;
    *_message->info_struc.typala=TYP;
    memcpy(buf ,comment,strlen(comment));
    memcpy(_message->comment,buf,20);
    _message->info_struc.lginfe=26;
    memcpy(buf1 ,info_,strlen(info_));
    memcpy(_message->info_struc.infoe ,buf1,26);
    _message->lginf=sizeof(_message->info_struc);
    *_message->info_struc.chaine=chaine;
        char buf1, buf2[256];
        buf2[0]=*_message->info_struc.chaine;
        buf2[1]='\0';
        IAS_TRACE("IAS_Ala_Spon_Send::setIAS_Ala_Spon_Send str"<<chaine
<<"msg.chaine="<<buf2);
        memcpy(buf2,_message->comment,20);
        buf2[20]='\0';
        IAS_TRACE("IAS_Ala_Spon_Send::setIAS_Ala_Spon_Send comment"<<comment
<<"msg.comment="<<buf2);
        memcpy(buf2,_message->info_struc.infoe,26);
        buf2[26]='\0';
        IAS_TRACE("IAS_Ala_Spon_Send::setIAS_Ala_Spon_Send info"<<info_
<<"msg.info="<<buf2);
    return;
}
//-------------------------------------------------------------------------
//  core dump
//-------------------------------------------------------------------------
string IAS_Ala_Spon_Send::dump ()
{
    string str ;
    char ligne[80];

    str = "\nsend alarm spon\n-----------\n";  str += INT_PROTmessage::dump();
    str += "Data\n----\n";
```

```
    if (_message->info_struc.chaine) {
      str+="chaine: ";
      str+=*_message->info_struc.chaine;
      str+="\n";
    }
    sprintf(ligne, "comment:              \n", _message->comment, _message-
>info_struc.infoe);  str += ligne;
    return str;
}
```

Alarm_Management.cpp

```
//-------------------------------------------------------------------------------
// AUTHOR  : Serafin F
// PROJECT : OMNeT++ Alarm Analyzer
// DOMAIN  : Gestiunea alarmelor
//-------------------------------------------------------------------------------
@file Alarm_Management.cpp
@brief gestiune alarme
//-------------------------------------------------------------------------------
#include "EventManagementC.r"
#include "DeferredSpontaneousSubscript  onImpl.hh"
#include "ALARM_ANALYZER_Configurator.h"
#include "ALARM_ANALYZER_COMMON.hh"
#include "SpondiffService.hh"
#include "SPONDIFF_Service_Task.hh"
#include "SPONDAlarm_AnalyzerClientFactory.  "
#include "SPONDAlarm_AnalyzerClient.hh"
#include "OmaIdlException.hh"
#include "OMNETErrorCodeC.h"
#include "OMNETFrameIdC.h"
#include "Frame.hh"
#include "CommonError.hh"
#include "TemporaryDifferedHandle.hh"
#include "ActlErrorCodeC.h"
#include "OmaComErrorCodeC.h"
#include "SPONErrno.hh"
#include "platformmapapi.h"

#define DeferredSpontaneousSubscriptionImpl_Methode_Number 2

//-------------------------------------------------------------------------------
// constructor
//-------------------------------------------------------------------------------
DeferredSpontaneousSubscriptionImpl::DeferredSpontaneousSubscriptionImpl()
{
        try
        {
        SPONDIFF_TRACE("DeferredSpontaneousSubscriptionImpl::Deferr  edSpo
onImpl(): start interface " << this );

        if (init_i() == -1)
                {
                OmaSard::OmaIdlException exception(OMNET::SERVICE_NOT_STARTED);
                exception.addFrame(OMNET::OMNET_SERVICE_NOT_STARTED_FRAME_ID);
                OmaSard::StringFrameAttribute serviceName(SPONDIFF_SERVICE_NAME);
                exception.addAttributeToLastFrame(serviceName);


        SPONDIFF_TRACE("DeferredSpontaneousSubscriptionImpl::DeferredSpon
onImpl() " \ << " : servicial nu este pornit, eroarea : "  <<
exception.getStringDescription());
                throw exception.getPlatformError();
                } // error internal initialization
        } // try
        catch(Platform::Error error)
                {
        SPONDIFF_TRACE("DeferredSpontaneousSubscriptionImpl::      eferredSpo
onImpl() Platform::Error " << error);
                throw error;
                }
```

```
        catch (SystemException)
                {
        SPONDIFF_TRACE("DeferredSpon
mplImpl : SystemException ");
                throw OmaSard::ComSystemError();
                }
} // DeferredSpontaneousSubscriptionImpl()

//-------------------------------------------------------------------------
// init_i() initializare interna
//-------------------------------------------------------------------------
int DeferredSpontaneousSubscriptionImpl::init_i(void)
{
int ret = 0;
        // client not initialized, ChronicleSubscription() not received
        _SPONDAlarm_AnalyzerClient = NULL;

        // initialize Client Factory ptr.
        ALARM_ANALYZER_Service_Object *spondService;
        spondService =
ALARM_ANALYZER_Service_Object::GetTheServiceObject(SPONDIFF_SERVICE_NAME);

        if (spondService == NULL)
                {
                // severe error, exception to be send to caller
                ret = -1;
                }
        else
                {
                _SpondService_Task = (SPONDIFF_Service_Task*) spondService -
>getServiceTask();
                if (_SpondService_Task == NULL)
                        {
                        // severe error, exception to be send to caller
                        ret = -1;
                        }
                else
                        {
                        // get service identity label
                        _identServ =_SpondService_Task -> getServiceINT_PROTId();
        _SPONDAlarm_AnalyzerClientFactory = _SpondService_Task ->  getClientFactory();
                        if (_SPONDAlarm_AnalyzerClientFactory == NULL)
                                // severe error, exception to be send to caller
                                ret = -1;
                        } // else ChronicleServiceTask valued
                } // else spondService valued
        return ret;
} // init_i()

//-------------------------------------------------------------------------
// destructor
//-------------------------------------------------------------------------
DeferredSpontaneousSubscriptionImpl::~DeferredSpontaneousSubscriptionImpl()
{
int ret;
        SPONDIFF_TRACE("DeferredSpontaneousSubscriptionImpl::DeferredSpontaneousSubscriptio
nImpl(): destructor " << this );

        if (_SPONDAlarm_AnalyzerClient != NULL)
                {
                // remove client
                ret = deleteClient_i   (&_SPONDAlarm_AnalyzerClient);
                }
} // ~DeferredSpontaneousSubscriptionImpl()

//-------------------------------------------------------------------------
// TEMPORARY subscription request, EXTERNAL INTERFACE
//-------------------------------------------------------------------------

void DeferredSpontaneousSubscriptionImpl::ChronicleSubscription (const OMNET::Rights
editionClasses,
```

```
OMNET::DeferredSpontaneous::EventCallBack_ptr callBack, OMNET::Invoke_Id invokeId, const
char * systemId)
        ACE_THROW_SPEC
((CORBA::SystemException,OMNET::Chronicle::Petri_Subscription_Invoke_Error,
Platform::Error))
{
        int ret;
        try
        {
        SPONDIFF_TRACE("DeferredSpontaneousSubscriptionImpl                    this
" << this << " impl,  invokeId=" << invokeId
        << " systemId=" << systemId);
        // rights control
        PLATFORM_ACCESS_CONTROL_BEGIN_METHOD(ChronicleSubscription);
        // test IOR receptionat
        OMNET::DeferredSpontaneous::EventCallBack_ptr test =
OMNET::DeferredSpontaneous::EventCallBack::_narrow(callBack);
        if (CORBA::is_nil(test))
        {
                CORBA::release(test);
                // IOR receptionat este eronat
                // LOG
                ALARM_ANALYZER_LOG_WARNING (ERRNO_ALARM_ANALYZER_SPON_BAD_IOR,
received in ChronicleSubscription method" );
                // emission d'une exception
                OmaSard::SymbolicFrameAttribute paramId("          ");
                OmaSard::StringFrameAttribute value("");
                OmaSard::ComBadParameterValue except(paramId, value);
        SPONDIFF_TRACE("erisie exception : " << except.getStringDescription());
        throw OmaIdlException::GetPlatformError(except);
        }
        CORBA::release(test);
        if (_SPONDAlarm_AnalyzerClient == NULL)
                {
                // check INT_PROT session     opened ok
                if (_SpondService_Task -> getINT_PROTsessionStateES() != true)
                        {
        OmaSard::OmaIdlException exception(OMNET::SERVICE_NOT_STARTED);
        exception.addFrame(OMNET::OMNET_SERVICE_NOT_STARTED_FRAME_ID);
        OmaSard::StringFrameAttribute serviceName(SPONDIFF_SERVICE_NAME);
        exception.addAttributeToLastFrame(serviceName);
        SPONDIFF_TRACE("DeferredSpontaneousSubscriptionImpl::DeferredSpontaneous          t
onImpl() " \ << " : INT_PROT NOT ES exception : " << exception.getStringDescription());
                throw exception.getPlatformError();
                        } // Int_Prot session not opened ok
                // initial call, create client
ret = createClient_i(  &_SPONDAlarm_AnalyzerClient, editionClasses, true, callBack);
        }
else
        {
        // next calls, update client
ret = updateClient_i   (&_SPONDAlarm_AnalyzerClient, editionClasses, true, callBack);
        }
if (ret == -1)
        {
        OmaSard::OmaIdlException exception(OMNET::INTERNAL_ERROR);
        exception.addFrame(OMNET::OMNET_INTERNAL_ERROR_FRAME_ID);
        OmaSard::StringFrameAttribute serviceName(SPONDIFF_SERVICE_NAME);
        exception.addAttributeToLastFrame(serviceName);
        SPONDIFF_TRACE("DeferredSpontaneousSubscriptionImpl::DeferredSpontaneous          t
onImpl() " \ << " : internal error exception : " << exception.getStringDescription());
                throw exception.getPlatformError();
                }
        else
                {
                return;
                }
        } // try
        catch(Platform::Error error)
                {
```

```
        SPONDIFF_TRACE("DeferredSpontaneousSubscriptionImpl::DeferredSpontaneousSubscriptionImpl
                " << error);
                throw error;
                }
}    ChronicleSubscription()

//-------------------------------------------------------------------------
// getSubscriptionInfo() : query drepturi abonament
//-------------------------------------------------------------------------
::OMNET::Rights_slice * DeferredSpontaneousSubscriptionImpl::getSubscriptionInfo ()
                    ACE_THROW_SPEC ((CORBA::SystemException, Platform::Error))
{
// control drepturi
    PLATFORM_ACCESS_CONTROL_BEGIN_METHOD(getSubscriptionInfo);
    try
    {
    ::OMNET::Rights_var rights = new OMNET::Rights ;
        SPONDIFF_TRACE("DeferredSpontaneousSubscriptionImpl::getSubscriptionInfo
for " << this << " impl, client is " << _SPONDAlarm_AnalyzerClient );
        // citire drepturi de la client si actualizare str
        if (_SPONDAlarm_AnalyzerClient != NULL)
                {
                (_SPONDAlarm_AnalyzerClient -> getEditionClasses()) ->
getOMNETRights(rights.inout());
                SPONDIFF_TRACE("DeferredSpontaneousSubscriptionImpl::getSubscriptionInfo
for client is " << _SPONDAlarm_AnalyzerClient << " , getEditionClasses() is " \
                        << (_SPONDAlarm_AnalyzerClient -> getEditionClasses()) -> dump() );
                } // client creat
        else
                {
                OmaSard::OmaIdlException exception(OMNET::NO_SUBSCRIPTION);
                exception.addFrame(OMNET::NO_SUBSCRIPTION_FRAME_ID);
                OmaSard::StringFrameAttribute serviceName(SPONDIFF_SERVICE_NAME);
                exception.addAttributeToLastFrame(serviceName);
        SPONDIFF_TRACE("DeferredSpontaneousSubscriptionImpl::DeferredSpontaneousSubscriptionI
mpl    " \ << " : internal error occurred : " << exception.getStringDescription());
                throw exception.getPlatformError();
                }
     return rights._retn();
     } // try
        catch(Platform::Error error)
                {
        SPONDIFF_TRACE("DeferredSpontaneousSubscriptionImpl                    << error);
                throw error;
                }
} // getSubscriptionInfo()

//-------------------------------------------------------------------------
//      createClient_i() metoda interna ptr creare client
//-------------------------------------------------------------------------
#define LOGMACROcreateClient_i(ret, fct)     {\
        SPONDIFF_TRACE("DeferredSpontaneousSubscriptionImpl::createClient_i " << this \
        << " ERROR, invalid return value from " << fct << " is "   << ret  );\
        ALARM_ANALYZER_LOG_ERROR_CPLT(ERRNO_ALARM_ANALYZER_SPON_CREATE_CLIENT,
"DeferredSpontaneousSubscriptionImpl::createClient_i () " \ << this << "
return value from   " << fct << " is "   << ret  ); \}
int DeferredSpontaneousSubscriptionImpl::createClient_i(SPONDAlarm_AnalyzerClient **
pSPONDAlarm_AnalyzerClient_, const OMNET::Rights editionClasses,
        OMNET::DeferredSpontaneous::Deferred_Subscription deferred,
        OMNET::DeferredSpontaneous::EventCallBack_ptr callBack)
{
int ret = 0;
        SPONDIFF_TRACE("DeferredSpontaneousSubscriptionImpl::createClient_i" );
        ret =  _SPONDAlarm_AnalyzerClientFactory -> createClient(
        SPONDAlarm_AnalyzerClient::TEMPORARY_SUBSCRIBER_TYPE
                                                        ,       pSPONDAlarm_AnalyzerClient_
                                                        ,       editionClasses
                                                        ,       deferred
                                                        ,       callBack
                                                        ,
        ALARM_ANALYZERClientRefTypeUNUSED
```

```
                                                        ,       false
                                                        ,       false
                                                        ,
        ActlImpl::InterfaceImpl::getSessionName().data()
                                                        );
SPONDIFF_TRACE("DeferredSpontaneousSubscriptionImpl::createClient() "
<< ActlImpl::InterfaceImpl::getSessionName().data() );
        switch (ret)
                {
                case -3 :       // maximum nb total subscribers
                        {
                LOGMACROcreateClient_i(ret , " _SPONDAlarm_AnalyzerClientFactory->
createClient() , maximum number of clients reached " )
                        // throw exception
                        OMNET::DeferredSpontaneous::Petri_Output_Right_Update_Error except;
                        except.infos = OMNET::DeferredSpontaneous::MORUEI_REJECTED;
                        throw except;
                        ret = 0;
                        break;
                        }
                case -2 :       // maximum nb permanent subscribers
                case -1:        // internal error
                        {
                        LOGMACROcreateClient_i(ret , " _SPONDAlarm_AnalyzerClientFactory->
createClient()  " )
                        // exception sent by caller
                        ret =  -1; // internal error exception
                        break;
                        }
                case 0 :
                        {
        SPONDIFF_TRACE("DeferredSpontaneousSubscriptionImpl::createClient() " << this \
        << " client created, ref is : " << (*pSPONDAlarm_AnalyzerClient_ ) ->
        getReference() << " address : " <<  (*pSPONDAlarm_AnalyzerClient_)  );
                break;
                        }
                default:
                        {
                        // conditie initiala
        LOGMACROcreateClient_i(ret , " _SPONDAlarm_AnalyzerClientFactory->
createClient()  bad return value " )
                        ret =  -1; // internal error exception
                        break;
                        }
                } // switch
        SPONDIFF_TRACE("DeferredSpontaneousSubscriptionImpl::createClient() " );
return ret;
} // createClient_i()

//-------------------------------------------------------------------------
//  updateClient_i() metoda interna ptr permanent client updating
//-------------------------------------------------------------------------
#define LOGMACROupdateClient_i(ret, fct)      {\
                SPONDIFF_TRACE("DeferredSpontaneousSubscriptionImpl::updateCl" <<
this \ << " ERROR, invalid return value from " << fct " is "   << ret  ); \
ALARM_ANALYZER_LOG_ERROR_CPLT(ERRNO_ALARM_ANALYZER_SPON_UPDATE_CLIENT,
"SPONDAlarm_AnalyzerClientFactory::updateClient ()" << this \
        << " ERROR, invalid return value from "  << fct " is "   << ret ); \}

int DeferredSpontaneousSubscriptionImpl::updateClient_i      (
        SPONDAlarm_AnalyzerClient ** pSPONDAlarm_AnalyzerClient_,
        const OMNET::Rights editionClasses,
        OMNET::DeferredSpontaneous::Deferred_Subscription deferred,
        OMNET::DeferredSpontaneous::EventCallBack_ptr callBack)
{
int ret = 0;
        SPONDIFF_TRACE("DeferredSpontaneousSubscriptionImpl::updateClient()  " );
        ret = _SPONDAlarm_AnalyzerClientFactory ->updateClient      (
        SPONDAlarm_AnalyzerClient::TEMPORARY_SUBSCRIBER_TYPE,
        pSPONDAlarm_AnalyzerClient_,
```

```
        editionClasses,
        deferred,

        callBack,
        (*pSPONDAlarm_AnalyzerClient_) -> getReference() );
        switch (ret)
                {
                case -3: // client gasit dar deja in modificare
                        {
                LOGMACROupdateClient_i(ret, "
client found but already UNDER MODIFICATION BY ANOTHER PROC " )
                        Platform::Error error;
                        error.code  = Platform::OMA_COMNET_CONCURRENT_ACCESS_PROHIBITED;
                        throw error;
                        break;
                        }
                case -2: // client not found
                        {
                LOGMACROupdateClient_i(ret, "
client NOT found   " )
                        // exception throw error by caller
                        ret = -1;
                        break;
                        }
                case -1 : // fatal error
                        {
        LOGMACROupdateClient_i(ret, "                                    )
                        // exception will be sent by caller
                        break;
                        }
                case 0:
                        {
                        break;
                        }
                default:  // unkwown value from method (internal error)
                        {
        LOGMACROupdateClient_i(ret, "                                    )
                        ret = -1;
                        break;
                        } // default
                } // switch
        SPONDIFF_TRACE("                                    " );
return ret;

} // updateClient_i()

//-------------------------------------------------------------------------
/ deleteClient_i() metoda interna ptr permanent client deletion
//-------------------------------------------------------------------------
#define LOGMACROdeleteClient_i(ret, fct)     {\

        SPONDIFF_TRACE("                                    " << this \
        << " ERROR, invalid return value from " << fct " is "   << ret  ); \
        ALARM_ANALYZER_LOG_ERROR_CPLT(ERRNO_ALARM_ANALYZER_SPON_REMOVE_CLIENT_RETRIEVE,
"                                    " << this \
        << " ERROR, invalid return value from "  << fct " is "   << ret ); \}

int DeferredSpontaneousSubscriptionImpl::deleteClient_i     (
        SPONDAlarm_AnalyzerClient ** pSPONDAlarm_AnalyzerClient_)
{
int ret=0;

        SPONDIFF_TRACE("                                    " );
        // calls factory method which does all the work, including client deletion
        // the identification key to get the client is its reference
        ret = _SPONDAlarm_AnalyzerClientFactory -> removeALARM_ANALYZERClient (
        pSPONDAlarm_AnalyzerClient_   ,
        SPONDAlarm_AnalyzerClient::TEMPORARY_SUBSCRIBER_TYPE ,
        (*pSPONDAlarm_AnalyzerClient_) -> getReference()
                                                        );
        switch (ret)
```

```
                case -3: // client gasit dar deja under modif by another impl
                        {
        LOGMACROdeleteClient_i(ret, "                    A
already UNDER MODIFICATION BY ANOTHER          " );
                Platform::Error error;
                error.code  = Platform::OMA_COM_CONCURRENT_ACCESS_PROHIBITED;
                throw error;
                break;
                        }
                case -2: // client not found
                        {
                        LOGMACROdeleteClient_i(ret, "  removeALARM_ANALY
client NOT found   " ); // exception throw error by caller
                        ret = -1;
                        break;
                        } // case -2
                case -1: // internal error
                        {
                        LOGMACROdeleteClient_i(ret, "          A
internal error   " ); // exception sent by caller
                        break;
                        }

                case 0: // ok client removed
                        {
                        // removal has been done
                        (*pSPONDAlarm_AnalyzerClient_) = NULL;

                        break;
                        } // case 0
                default:  // unkwown value from method (internal error)
                        {
                        LOGMACROdeleteClient_i(ret, "Unknown return code
removeALARM_ANALYZERClient () " );

                        ret = -1;
                        break;
                        }
        } // switch
        SPONDIFF_TRACE("DeferredSpontaneousSubscriptionImpl::deleteClient      END  " );
return ret;

} // deleteClient_i()

//-------------------------------------------------------------------------------
// creator de implementare
//-------------------------------------------------------------------------------
ActlImpl::InterfaceImpl* DeferredSpontaneousSubscriptionImpl::CreateImplInstance()
{
    ActlImpl::InterfaceImpl *res;

    res = new DeferredSpontaneousSubscriptionImpl();
    SPONDIFF_TRACE("DeferredSpontaneousSubscriptionImpl::CreateImplInstance
Create :  " << res );
    return res;
}
//-------------------------------------------------------------------------------
// metoda generala
//-------------------------------------------------------------------------------
string*  DeferredSpontaneousSubscriptionImpl::GetMethodes()
{
   string *methodes;
   methodes = new string[DeferredSpontaneousSubscriptionImpl_Methode_Number];
   methodes[0] = "ChronicleSubscription";
   methodes[1] = "getSubscriptionInfo";
   return methodes;
}
//-------------------------------------------------------------------------------
// numărul de metode
//-------------------------------------------------------------------------------
int DeferredSpontaneousSubscriptionImpl::GetMethodesNumber()
```

```
{
   return DeferredSpontaneousSubscriptionImpl_Methode_Number;
}
```

Petri_Automaton.cpp

```
//-------------------------------------------------------------------------------
// AUTHOR  : Serafin P
// PROJECT : OMNeT++ Alarm Analyzer
// DOMAIN  : Petri Automaton
//-------------------------------------------------------------------------------
@file PETRI_Automaton.cpp
@brief Asamblare retea Petri
//-------------------------------------------------------------------------------
#ifndef __PETRI_Automaton_cc__
#define __PETRI_Automaton_cc__

#include "PETRI_Automaton.hh"
#include "INT_PROTmessage.hh"
#include "ALARM_ANALYZERMacros.hh"
#include "CHRONICLEErrno.hh"
#include "INT_PROTErrno.hh"
//-------------------------------------------------------------------------------
//constructor
//-------------------------------------------------------------------------------
PETRI_Automaton::PETRI_Automaton()
{
    _state = "IDLE";
}
//-------------------------------------------------------------------------------
//destructor
//-------------------------------------------------------------------------------
PETRI_Automaton::~PETRI_Automaton()
{
}
//-------------------------------------------------------------------------------
// automaton event : ChronicleCommand
//-------------------------------------------------------------------------------
int PETRI_Automaton::ChronicleCommand(void)
{
    if (_state == "IDLE") {
        _state = "RUNNING";
        return (int)1;
    } else {
       ALARM_ANALYZER_LOG_NOTICE(ERRNO_ALARM_ANALYZER_UNEXPECTED_CORBA_MES,
"ChronicleCommand out of sequence");
       return (int)-1;
    }
}
//-------------------------------------------------------------------------------
// automaton event : ChronicleExtraInfoReply
//-------------------------------------------------------------------------------
int PETRI_Automaton::ChronicleValidationReply(void)
{
    if (_state == "WAIT_VALIDATION") {
        _state = "RUNNING";
        return (int)1;
    } else {
       ALARM_ANALYZER_LOG_NOTICE(ERRNO_ALARM_ANALYZER_UNEXPECTED_CORBA_MES,
"ChronicleExtraInfoReply out of sequence");
       return (int)-1;
    }
}
//-------------------------------------------------------------------------------
// automaton event : ChronicleExtraInfoReply
//-------------------------------------------------------------------------------
int PETRI_Automaton::ChronicleExtraInfoReply(void)
{
    if (_state == "WAIT_ADDITIONAL_LINE") {
        _state = "RUNNING";
```

```
        return (int)1;
    } else {
       ALARM_ANALYZER_LOG_NOTICE(ERRNO_ALARM_ANALYZER_UNEXPECTED_CORBA_MES,
"ChronicleExtraInfoReply out of sequence");
        return (int)-1;
    }
}
//-------------------------------------------------------------------------------
// automaton event : ChronicleCancel
//-------------------------------------------------------------------------------
int PETRI_Automaton::ChronicleCancel(void)
{
    if (_state != "IDLE") {
        _state = "IDLE";
        return (int)1;
    } else {
       ALARM_ANALYZER_LOG_NOTICE(ERRNO_ALARM_ANALYZER_UNEXPECTED_CORBA_MES,
"ChronicleCancel out of sequence");
        return (int)-1;
    }
    return (int)1;
}
//-------------------------------------------------------------------------------
// check : check the automaton sequencing according the spcc (message identifier)
//-------------------------------------------------------------------------------
int PETRI_Automaton::check (long spcc_)
{
    switch (spcc_) {
    case CCHFRP:                // CHRONICLE Final Report
    case CCHACC:                // CHRONICLE Acceptance
    case CCHCRJ:                // CHRONICLE Reject
        if (_state != "IDLE") {
            _state = "IDLE";
            return (int)1;
        } else {
        ALARM_ANALYZER_LOG_NOTICE(ERRNO_ALARM_ANALYZER_UNEXPECTED_NMT_MES,
"NT_PROT receptionnat out of sequence : "<<hex<<spcc_<<dec);
        PETRI_TRACE("out of sequence : " << spcc_ <<" " <<dump());
            return (int)-1;
        }
    case CCHPRP:                // CHRONICLE Partial Report
        if (_state == "RUNNING") {
            return (int)1;
        } else {
       ALARM_ANALYZER_LOG_NOTICE(ERRNO_ALARM_ANALYZER_UNEXPECTED_NMT_MES,
"PETRI_Partial_Report out of sequence : "<<hex<<spcc_<<dec);
       PETRI_TRACE("PETRI_Partial_Report out of sequence : " << spcc_ <<" " <<dump());
            return (int)-1;
        }
    case CCHVRQ:                // CHRONICLE Validation Request
        if (_state == "RUNNING") {
            _state = "WAIT_VALIDATION";
            return (int)1;
        } else {
       ALARM_ANALYZER_LOG_NOTICE(ERRNO_ALARM_ANALYZER_UNEXPECTED_NMT_MES,
"Petri_Validation_Request out of sequence : "<<hex<<spcc_<<dec);
       PETRI_TRACE("Petri_Validation_Request out of sequence : " << spcc_ <<" "
<<dump());
            return (int)-1;
        }
    case CCHARQ:                // CHRONICLE Additional Line Request
        if (_state == "RUNNING") {
            _state = "WAIT_ADDITIONAL_CHRONICLE";
            return (int)1;
        } else {
       ALARM_ANALYZER_LOG_NOTICE(ERRNO_ALARM_ANALYZER_UNEXPECTED_NMT_MES,
"Petri_Additional_Line_Request out of sequence : "<<hex<<spcc_<<dec);
       PETRI_TRACE("Petri_Additional_Chronicle_Request out of sequence : " << spcc_ <<" "
<<dump());
            return (int)-1;
        }
```

```
    default :
    ALARM_ANALYZER_LOG_WARNING(ERRNO_ALARM_ANALYZER_UNKNOWN_MES, "                    ");
    return (int)-1;
    }
}
//-------------------------------------------------------------------------------
// dump
//-------------------------------------------------------------------------------
string PETRI_Automaton::dump(void)
{
    string str = "";
    str += " CHRONICLE automaton state = ";   str += _state;   str += ".";
    return str;
}
#endif
```

BIBLIOGRAFIE

[1] *** Alcatel 1000 E10 (OCB283) – *Description du système de commutation*, IFA, Institut de Formation Alcatel, Lannion, Franţa, 1995

[2] *** Alcatel 1000 E10 (OCB283) – *Gestion des alarmes*, IFA, Institut de Formation Alcatel, Lannion, Franţa, 1996

[3] Armen Aghasaryan – *Formalisme Hidden Markov Model pour les réseaux de Petri partiellement stochastiques. Application au diagnostique de pannes dans les systemes répartis*, Teză de doctorat nr.2145, Universitatea din Rennes I, Franţa, 1998

[4] Armen Aghasaryan, Renée Boubour, Eric Fabre, Claude Jard, Albert Benveniste – *Une approche par réseaux de Petri aux problèmes de détection de panne et de diagnostic dans les systèmes distribués*, Raport de cercetare nr. 1117, IRISA, Institut de Recherche en Informatique et Systèmes Aléatoires, Rennes, Franţa, 1997

[5] Armen Aghasaryan, Christophe Dousson, Eric Fabre, Aomar Osmani, Yannick Pencolé – *Modeling Fault Propagation in Telecommunications Networks for Diagnosis Purposes*, Proceedings, al 18-lea Simpozion WTC, World Telecommunications Congress, Paris, Franţa, 2002

[6] Armen Aghasaryan, Christophe Dousson – *Mixing Chronicle and Petri Net Approaches in Evolution Monitoring Problems*, Proceedings, a 12-a Conferinţă WPD, Workshop Principles of Diagnosis, pag. 1-7, San Sicario, Italia, Martie, 2001

[7] Armen Aghasaryan – *Diagnostic de pannes dans les grands systèmes répartis: une approche à base de Réseaux de Pétri partiellement stochastiques*, Seria Calculateurs Parallèles, nr.11, pag. 447-462, Editura Hermès-Science, 1999

[8] Rakesh Agrawal, Ramakrishnan Srikant - *Fast algorithms for mining association rules in large databases*, Proceedings, a 20-a Conferinţă VLDB, Very Large Data Bases, pag. 478-499, Santiago de Chile, Chile, 1994

[9] Rakesh Agrawal, Tomasz Imielinski, Arun Swami - *Mining association rules between sets of items in large databases*, Proceedings, ACM SIGMOD Conference on Management of Data, 1993.

[10] Jose Aguilar, Kouamana Bousson, Christophe Dousson, Malik Ghallab, Anthony Guasch, Rob Milne, Charlie Nicol, Jose Quevedo, Louise Travé-Massuyès – *TIGER: real-time situation assessment of dynamic systems*, Intelligent Systems Engineering, Volumul 3(3), pag. 103-124, Octombrie, 1994

[11] Alfred V. Aho - *Algorithms for finding patterns in strings*. Handbook of Theorical Computer Science, pag. 256-300, Editura Elsevier, 1990

[12] James F. Allen - *Maintaining Knowledge about Temporal Intervals*. Seria Communication of ACM, nr. 26, pag. 123-154, 1983

[13] James F. Allen - *Towards a General Theory of Action and Time*. Seria Artificial Intelligence, nr.23, pag.72-80, 1984

[14] James F. Allen - *A Common-Sense Theory of Time*. Proceedings, IJCAI, nr.9, pag. 528-531, 1985.

[15] James F. Allen, Philip J. Hayes - *Moments and Points in an Interval-based Temporal-based Logic*. Seria Computational Intelligence, nr.5, pag. 225-238, 1989

[16] François Baccelli, Guy Cohen, Geert Jan Olsder, Jean-Pierre Quadrat – *Synchronisation and Linearity: an algebra for discrete event systems*, Editura John Wiley&Sons, Seria Probability and Mathematical Statistics, http://www-rocq.inria.fr/metalau/cohen/SED/book-online.html, 1992

[17] Peter van Beek - *Approximation Algorithms For Temporal Reasoning.* Proceedings, IJCAI, nr.11, pag. 1291-1296, 1989

[18] Peter van Beek - *Reasoning About Qualitative Temporal Information*, Seria Artificial Intelligence, nr. 58, pag. 297-326, 1992

[19] Albert Benveniste, Bernard C. Lévy, Eric Fabre, Paul Le Guernic – *A Calculus of Stochastic Systems: Specification, Simulation and Hidden State Estimation*, Journal of Theoretical Computer Science, vol.152, nr.2, pag. 171-217, 1995

[20] Albert Benveniste, Stefan Haar, Eric Fabre, Claude Jard – *Fault Diagnosis for Distributed Asynchronous Dynamically Reconfigured Discrete Event Systems*, Proceedings, al 16-lea Congres IFAC, Praga, Cehia, Iulie 2005

[21] Serge Bibas, Marie-Odile Cordier, Philippe Dague, François Lévy, Laurence Rozé - *Scenario generation for telecomunication network supervision.* Workshop on AI in Distributed Information Networks, Montreal, Canada, 1995

[22] Serge Bibas, Marie-Odile Cordier, Philippe Dague, Christophe Dousson, François Lévy, Laurence Rozé – *Alarm driven supervision for telecommunication networks: on-line scenario generation.* Annals of Telecommunications nr.9/10, Tome 51, pag. 493-500, Octombrie 1996

[23] Serge Bibas, Marie-Odile Cordier, Philippe Dague, Christophe Dousson, François Lévy, Laurence Rozé – *Using a model-based approach for monitoring telecommunication networks : the GASPAR system,* IEEE Colloquium on AI for Network Management Systems, pag. 51-52, Aprilie 1997

[24] Claudia Böttcher, Oskar Dressler - *A framework for controlling model-based diagnosis systems with multiple actions,* Annals of Mathematics and Artificial Intelligence, special issue on Model-Based Diagnosis, nr. 11(1-4), Editura J.C. Baltzer AG, pag. 241-261, 1994.

[25] Renée Boubour – *Suivi des pannes par corrélation causale d'alarmes dans les systèmes répartis*, Teza de doctorat, Universitatea din Rennes I, Franţa, 1997

[26] Renée Boubour, Claude Jard – *Fault Detection in Telecommunication Networks Based on a Petri Net Representation of Alarm Propagation,* Proceedings, ICATPN, 1997

[27] Kouamana Bousson, Laurent Zimmer, Louise Travé-Massuyès - *Causal Model-Based Diagnosis of Dynamic Systems*, al 5-lea International Workshop Principles of Diagnosis, New Paltz, New York, SUA, 1994.

[28] Bertham C. Bruce - *A Model for Temporal References and Application in a Question Answering Program*, Seria Artificial Intelligence, nr.3, pag. 1-25, 1972

[29] Sylvie Cauvin, Marie-Odile Cordier, Christophe Dousson, Philippe Laborie, François Lévy, Jacky Montmain, Marc Porcheron, Isabelle Servet, Louise Travé-Massuyès – *Monitoring and Alarm Interpretation in Industrial Environments*, Seria AI Communications vol.11 (3-4), pag. 139-173, IOS Press, 1998

[30] CCITT – *Livre Bleu. Tome II – Fascicule II.3 Qualité du service, gestion du réseau et ingénierie du trafic.* Recommandations E.401 – E.880, IX Assemblée Plénière, Melbourne, Canada, 1988

[31] Young-il Choo – *Concurrency Algebra and Petri Nets*, Raport tehnic 5190:TR:85, California Institute of Technology, SUA, 1985

[32] Marie-Odile Cordier, Christophe Dousson – *Alarm Driven Monitoring Based on Chronicles*, Proceedings, al 4-lea Simpozion Fault Detection Supervision & Safety for Technical Processes, pag.286-291, Budapest, Ungaria, Iunie, 2000

[33] Marie-Odile Cordier, Christine Largouët - *Using model-checking techniques for diagnosing discrete-event systems*, Proceedings, al 12-lea International Workshop on Principles of Diagnosis DX-01, Sansicario, Italia, 2001

[34] Marie-Odile Cordier, Sylvie Thiébaux - *Event-based diagnosis for evolutive systems*, Proceedings, al 5-lea International Workshop on Principles of Diagnosis DX-94, New Paltz, New York, SUA, 1994.

[35] Victor Croitoru – *Comunicaţii digitale : teorie şi experiment*, Editura Adisan, Bucureşti, 1995

[36] Victor Croitoru – *Reflecţii asupra evoluţiei învăţământului românesc de comunicaţii*, Comunicare la sesiunea *Istoria telecomunicaţiilor în România*, Academia Română, 15 Aprilie, 2003

[37] Adnan Darwiche - *Model-based diagnosis using structured system descriptions*, Journal of Artificial Intelligence Research nr.8, pag. 165-222, 1998.

[38] Thomas L. Dean, Drew V. McDermott - *Temporal database management*, Seria Artificial Intelligence nr.32, pag. 1-55, 1987

[39] Rami Debouk, Stéphane Lafortune, Demosthenis Teneketzis – *Coordinated Decentralized Protocols for Failure Diagnosis of Discrete Event Systems*, Seria Discrete Event Dynamic Systems, Vol. 10, nr.1-2, Editura Kluwer Academic, Ianuarie, 2000

[40] Rina Dechter, Itay Meiri, Judea Pearl - *Temporal constraint networks*. Artificial Intelligence Special Volume on Knowledge Reprezentation, pag. 61-95, Editura Elsevier, 1991

[41] Luc Dehaspe, Hannu Toivonen - *Frequent query discovery : unifying ILP aproach to association rule mining,* Raport Tehnic CW:258, Katholieke Universiteit Leuven, Belgia, 1998

[42] Virgil Dobrotă – *Reţele digitale în telecomunicaţii, Volumul I: Comutaţia digitală. Analiza traficului*, Ediţia a 3-a, Editura Mediamira, Cluj-Napoca, 2002

[43] Virgil Dobrotă – *Reţele digitale în telecomunicaţii. Volumul III : OSI şi TCP/IP*, Ediţia a 2-a, Editura Mediamira, Cluj-Napoca, 2003

[44] Christophe Dousson - *Alarm driven supervision for telecommunication networks : on-line chronicle recognition,* Annals of Telecommunications nr.9/10, Tome 51, pag.501-508 CNET France Telecom, Lannion, Franţa, Octombrie, 1996

[45] Christophe Dousson – *Extending and Unifying Chronicle Representation with Event Counters*, Proceedings, al 15-lea Simpozion ECAI, pag. 257-261, IOS Press, Lyon, Franţa, Iulie, 2002

[46] Christophe Dousson – *Suivi d'evolutions et reconnaissance de chroniques*, Teză de doctorat nr.1811, Universitatea Paul Sabatier Toulouse, Franţa, Septembrie, 1994

[47] Christophe Dousson, Paul Gaborit, Malik Ghallab - *Situation Recognition: Representation and Algorithms* Proceedings, al 13-lea IJCAI, pag. 166-172, Chambéry, Franţa, August, 1993

[48] Christophe Dousson, Thang Vu Dong – *Découverte de chroniques avec contraintes temporelles à partir de journaux d'alarmes. Application à la supervision de réseaux de télécommunications*, Acte 1ère Conférence d'Aprentissage (CAP), pag.17-24, Palaiseau - Paris, Franţa, Iunie, 1999

[49] Doina Drăgulescu, Mirela Toth-Taşcău, Petru L. Serafin – *Analiza spaţiului de lucru al unui robot RRTRRR prin metoda modelării statistice*, al XI-lea Simpozion Naţional de Roboţi Industriali, Vol.1, Lucrarea nr.10, Timişoara, Octombrie, 1992

[50] Oskar Dressler - *On-line diagnosis and monitoring of dynamic systems based on qualitative models and dependency-recording diagnosis engines*, Proceedings ECAI-96, pag. 481-485, Budapesta, Ungaria, 1996.

[51] Eric Fabre, Armen Aghasaryan, Albert Benveniste, Renée Boubour, Claude Jard – *Fault detection and diagnosis in distributed systems: an approach by partially stochastic Petri nets*, Journal of

Discrete Event Dynamic Systems, no.2, vol.8, pag. 203-231, Ed. Kluwer Academic, Boston, SUA, 1998

[52] Usama M. Fayyad, Heikki Mannila, Raghu Ramakrishnan - *Data Mining Knowledge Discovery*, Editorial, IEEE Expert Special Issue on Data Mining, Vol.8, nr.1, pag.5-6, Ianuarie 2004

[53] Françoise Fessant, Christophe Dousson, Fabrice Clérot – *Mining on a telecommunication alarm log to improve the discovery of frequent patterns*, Industrial Conference on Data Mining (ICDM), Leipzig, Germania, Iulie, 2003

[54] Françoise Fessant, Christophe Dousson, F. Clérot – *SOM-based pre-processing of a telecomunications alarm log for a more accurate discovery of recurrent phenomena*, International Conference on Neutral and Artificial Intelligence (ICNNAI), Minsk, Bielorusia, Noiembrie, 2003

[55] Anthony Galton - *A Critical Examination of Allen's Theory of Action and Time*, Artificial Intelligence, nr. 42, pag. 159-188, 1990

[56] Anthony Galton - *Logic for Information Technology*. Editura John Wiley & Sons Ltd., 1990.

[57] Georges Fiche, Gérard Hébuterne - *Trafic et performances des réseaux de télécoms*, Ed. Hermes-Science, Groupe des Ecoles de Télécommunications & Lavoisier, Paris, Franţa, 2003

[58] Bruno Guerraz, Christophe Dousson – *Chronicles Construction Starting from the Fault Model of the System to Diagnose*, International Workshop on Principles of Diagnosis (DX), pag.51-56, Carcassonne, Franţa, Iunie, 2004

[59] Bernhard Ganter - *Algoritmen zur formalen begriffsanalyse*, Beitrage zur Beigriffanalyse, Editura Wissenschaft-Verlag, 1987

[60] Alban Grastien, Marie-Odile Cordier, Christine Largouët – *Extending Decentralized Discrete-Event Modeling to Diagnose Reconfigurable Systems*, International Workshop on Principles of Diagnosis (DX), Carcassonne, Franţa, Iunie, 2004

[61] Malik Ghallab - *On chronicles : representation and on-line recognition*, Proceedings, a 5-a Conferinţă Internaţională Principles of Knowledge Representation and Reasoning (KRR), 1996

[62] Zonghua Gu, Kang G. Shin – *Analysis of Event-Driven Real-Time Systems with Time Petri Nets: A Translation-Based Approach,* DIPES 2002, Seria Distributed and Parallel Embedded Systems, pag. 31-40, Montreal, Canada, 25-30 August, 2002

[63] Larry Holloway, Deepa N. Pandalai – *Template Languages for Fault Monitoring of Timed Discrete Event Processes*, IEEE Transactions on Automatic Control, Nr.45(5), 2000

[64] Frank Höppner – *Learning dependencies in multivariate time series*, KDSTD Workshop, pag. 25-31, Lyon , Franţa, 2002

[65] Stephen D. Hudson, James C. Johnson, Umar Syyid – *The ACE Programmer's Guide*, Ed. Addison-Westley, Octombrie 2003

[66] Alimpie Ignea, Mircea Chivu, Ioan Borza – *Măsurări electrice şi electronice în instalaţii*, Editura Orizonturi Universitare, Timişoara, 1998

[67] Lucian Ioan – *Probabilităţi şi variabile aleatorii în telecomunicaţii – teorie şi aplicaţii*, Ed. Matrix Rom, Bucureşti, 1998

[68] Lucian Ioan, Grazziela Niculescu – *Elemente de ingineria traficului în telecomunicaţii – teorie şi aplicaţii*, Ed. Matrix Rom, Bucureşti, 2001

[69] Claude Jard – *Synthesis of distributed testers from true-concurrency models of reactive systems*, International Journal of IST, Editura Elsevier, 2002

[70] Gabriel Jakobson, Mark D. Weissman - *Real-time telecommunication netwok management extending event correlation with temporal constraints*, Proceedings, al 5-lea Simpozion ISNIM, pag. 290-301, 1995

[71] Brian Knight, Jixin Ma - *An Extended Temporal System Based on Points and Intervals*, Information System, nr.18, pag.111-120, 1993

[72] Philippe Laborie, Jean-Paul Krivine - *Automatic generation of chronicles and its application to alarm processing in power distribution systems*, Proceedings, al 8-lea International Workshop on Principles of Diagnosis DX'97, Mont Saint Michel, Franţa, 1997

[73] Philippe Laborie, Jean-Paul Krivine - *GEMO: A model-based approach for an alarm processing function in power distribution networks*, Proceedings, International Conference on Intelligent System Application to Power Systems ISAP-97, Seul, Coreea de Sud, 1997

[74] Peter Ladkin, Alexander Reinefeld – *Fast Algebraic Methods for Interval Constraint Problems*, Annals of Mathematical and Artificial Intelligence, vol.19, nr.3-4, pag. 383-411, 1997

[75] Peter Ladkin - *Effective Solution of Qualitative Interval Constraint Problems*, Artificial Intelligence, nr.52, pag.105-124, 1992

[76] Tony Lindeberg – *Discrete Scale-Space Theory and the Scale-Scale Primal Sketch*, Teză de doctorat, Departamentul Numerical Analysis and Computer Science, KTH Royal Institute of Technology, Stockholm, Suedia, August, 1998

[77] Michael Luxenburger - *Implications partielles dans un contexte*, Mathemathique, Informatique et Sciences Humaines, nr. 29, pag. 35-55, 1991

[78] Alan K. Mackworth, Eugene C. Freuder – *The complexity of constraint satisfaction revisited*, Artificial Intelligence, nr. 59, Elsevier, 1993

[79] Roberto Maiocchi - *Automatic Deduction of Temporal Information*, ACM Transactions on Database Systems, nr.4, pag. 647-688, Milano, Italia, 1992

[80] Heikki Manilla, Hannu Toivonen - *Discovering frequent episodes in sequences*, Proceedings, primul simpozion KDD, pag. 210-215, 1995

[81] Marco Ajmone Marsan, Gianfranco Balbo, Gianni Conte, Susanna Donatelli, Giuliana Franceschinis – *Modeling with Generalized Stochastic Petri Nets*, Editura John Wiley, 1995

[82] Emmanuel Mayer - *Apprentissage inductif de scenarios pour la supervision de reseaux de telecommunication* , Seminar la Universitatea din Rennes I, Franţa, Decembrie, 1999

[83] Drew McDermott - *A Temporal Logic for Reasoning about Processes and Plans*, Cognitive Science, nr. 6, pag.101-155, 1982

[84] Drew McDermott – *Mind and Mechanism*, Editura MIT Press, Ediţia I, Octombrie, 2001

[85] Drew McDermott –*A Critique of Pure Reason*, Computational Intelligence, nr. 3, pag.151-237, 1987

[86] Kenneth L. McMillan – *Symbolic model checking : an approach to the state explosion problem*, Computer Science, Carnegie Mellon University, SUA, 1993

[87] Tadao Murata - *Petri Nets: Properties, Analysis and Applications*, Proceedings of the IEEE, Vol. 77, Nr.4, pag. 541-580, Aprilie, 1989

[88] Miranda Naforniţă, Carmen Munteanu - *Comunicaţii de date*, Ed. Gh.Asachi, Iaşi, 1996

[89] Grazziela Niculescu, Ştefania Bărbălău – *Analiza şi modelarea sistemelor de comunicaţii*, Ed. Matrix Rom, Bucureşti , 1997

[90] Grazziela Niculescu, Lucian Ioan - *Tehnici şi sisteme de comutaţie*, Ed. Matrix Rom, Bucureşti, 2000

[91] Grazziela Niculescu – *Traficul în rețelele de telecomunicații*, Ed.Tehnică, București, 1994

[92] Morgens Nielsen, Gordon Plotkin, Glynn Winskel – *Petri nets, event structures and domains*, Theoretical Computer Science, nr.13, pag.85-108, Elsevier, 1980

[93] Yossi Nygate - *Event correlation using rule and object-based techniques*. Proceedings, al 4-lea ISNIM, pag. 279-289, Mai, 1995

[94] Marius Oteșteanu – *Sisteme de transmisie și comutație*, Editura Orizonturi Universitare, Timișoara, 2001

[95] Marius Oteșteanu, Florin Alexa, Cornel Balint – *Telefonia numerică. Sistemul Alcatel 1000 E10*, Editura de Vest, Timișoara, 2004

[96] Yannick Pencolé – *Diagnosability Analysis of Distributed Discrete Event Systems*, International Workshop on Principles of Diagnosis (DX), Carcassonne, Franța, Iunie, 2004

[97] Yannick Pencolé, Marie-Odile Cordier, Laurence Rozé - *Incremental decentralized diagnosis approach for the supervision of a telecommunication network*, working notes of the 12th International Workshop on Principles of Diagnosis DX-01, Sansicario, Italia, 2001.

[98] Yannick Pencolé, Marie-Odile Cordier - *A formal framework for the decentralized diagnosis of large scale discrete event systems and its application to telecommunication networks*, Artificial Intelligence Journal , Editura Elsevier, nr. 164 (1-2), pag. 121-170, 2005

[99] James L. Peterson - *Petri Net Theory and the Modeling of Systems*, Editura Prentice-Hall, New Jersey, SUA, 1981

[100]Carl Adam Petri - *Kommunikation mit Automaten*, Institut für Instrumentelle Mathematik, Schriften des IIM Nr.3, Universitatea Tehnică din Darmstadt, Germania, 1962

[101]Raymond Reiter - *A theory of diagnosis from first principles*, Artificial Intelligence nr.32(1), pag. 57-96, 1987.

[102]Tatiana Rădulescu – *Telecomunicații*, Editura Teora, București, 1998

[103]Tatiana Rădulescu – *Rețele de telecomunicații*, Editura Thalia, București, 2003

[104]Bill Roscoe – *The Theory and Practice of Concurrency*, Editura Prentice-Hall, Seria Computer Science, 1998

[105]Laurence Rozé - *Supervision of telecommunication network: A diagnoser approach*, Proceedings, al 8-lea International Workshop on Principles of Diagnosis DX'97, Mont Saint Michel, Franța, 1997.

[106]Carsten Rust, Franz J. Ramming – *A Petri Net Based Approach for the Design of Dynamically Modifiable Embedded Systems*, DIPES 2004, Distributed and Parallel Embedded Systems, Toulouse, Franța, 23-26 August, 2004

[107]Raja Sengupta, Stéphane Lafortune, Demosthenis Teneketzis - *Diagnosability of discrete-event systems*, IEEE Transactions Automatic Control nr. 40 (9), pag. 1555-1575, 1995.

[108]Raja Sengupta, Stéphane Lafortune, Demosthenis Teneketzis - *Failure diagnosis using discrete-event models,* IEEE Transactions Control Systems Technology nr. 4 (2), pag. 105-124, 1996.

[109]Douglas C. Schmidt – *An Object-Oriented Framework for Experimenting with Alternative Process Architectures for Parallelizing Communication Subsystems*, teză de doctorat, University of California, Irvine, SUA, 1995

[110]Douglas C. Schmidt, Burkhard Stiller, Tatsuya Suda, Martina Zitterbart - *Configuring Function-based Communication Protocols for Distributed Applications*, Proceeding, a 8-a Conferință Internațională Upper Layer Protocols, Architectures and Applications, pag. 1–13, Barcelona, Spania, 1994

[111]Douglas C. Schmidt, Stephen D. Huston – *C++ Network Programming : Mastering Complexity with ACE and Patterns*, Volumul 1, Seria C++ In-Depth Series, Bjarne Stroustrup, Editura Addison-Westley, Decembrie 2001

[112]Petru L. Serafin, François Mellouet – *Reconaissance automatique des formes par la logique floue*, Seminar de sinteză ESEO, Ecole Supérieure d'Electronique de l'Ouest, Angers, Franţa, 26-27 Iunie, 1997

[113]Petru L. Serafin – *Petri Net and Chronicle Recognition in Analysis of Telecommunication Alarm Logs*, Acta Tehnica Napocensis – Seria Electronics and Telecommunications (acceptată pentru publicare), Universitatea Tehnică din Cluj-Napoca, 2005

[114]Petru L. Serafin – *Algoritm pentru determinarea cronicilor frecvente în jurnalele de alarme ale reţelelor de telecomunicaţii*, Sesiunea de comunicări ştiinţifice *Doctor ETc*, Universitatea Politehnica Timişoara, Facultatea de Electronică şi Telecomunicaţii, Timişoara, 22 Septembrie, 2005

[115]Petru L. Serafin – *Network Simulation using OMNet++ environment*, Buletinul Ştiinţific al UPT, Tom 49 (63), Fascicola 1, pag. 407-411, Simpozionul Electronică şi Telecomunicaţii, Timişoara, 22-23 Octombrie, 2004

[116]Petru L. Serafin – *Performance Analysis of Packet-based Voice Networks*, Buletinul Ştiinţific al UPT, Tom 49 (63), Fascicola 1, pag. 412-418, Simpozionul Electronică şi Telecomunicaţii, Timişoara, 22-23 Octombrie, 2004

[117]Petru L. Serafin – *Quality Limitations for Packetized Voice Calls* – Buletinul Ştiinţific al UPT, Tom 45 (59), Fascicola 1, pag. 176-179, Simpozionul Electronică şi Telecomunicaţii, Timişoara, 23-24 Noiembrie, 2000

[118]Crişan Strugaru – *Sisteme de comunicaţii digitale* – Editura Orizonturi Universitare, Seria Tehnică, Timişoara, 2000

[119]Louise Travé-Massuyès, Kouamana Bousson - *Non causal vs. causal qualitative modelling and simulation*, Intelligent Systems Engineering nr.2 (3), pag. 159-182, 1993

[120]Louise Travé-Massuyès, Philippe Dague - *Le raisonnement qualitatif pour les sciences de l'ingénieur*, Editura Hermes, 1997

[121]Louise Travé-Massuyès, Robert Milne - *Application-oriented qualitative reasoning*, The Knowledge Engineering Review nr.10(2), 1995

[122]Louise Travé-Massuyès, Robert Milne - *TIGERTM: Gas turbines condition monitoring using qualitative model based diagnosis*, IEEE Expert Intelligent Systems & Applications nr.12 (3), pag. 21-31, 1997.

[123]Louise Travé-Massuyès, Robert Milne - *Gaps between research and industry related to model based and qualitative reasoning*, Proceedings, ECAI'98 workshop on "Model Based Systems and Qualitative Reasoning", Brighton, Marea Britanie, 1998.

[124]Louise Travé-Massuyès, Renaud Pons - *Causal ordering for multiple mode systems*, 11th Int. Workshop on Qualitative Reasoning about Physical Systems, Cortona, Italia, 1997

[125]Edward P. K. Tsang - *Time Structure for AI*, Proceedings of the IJCAI, nr.10, pag. 456-461, 1987

[126]András Varga - *OMNet++ User Manual*, Departamentul Telecommunicaţii, Universitatea Tehnică Budapesta, Ungaria, 2000

[127]András Varga - *OMNet++ Discrete Event Simulation System*, Proceeding of the European Simulation Multiconference ESM2001, Praga, Cehia, 6-9 Iunie, 2001

[128]András Varga - *Software Tool for Networking*, IEEE Network Interactive, vol.16 nr.4, Iulie 2002

[129]Marc B. Vilain - *A System for Reasoning about Time*. Proceedings la prima conferință AAAI, pag. 197-201, Pittsburgh, SUA, 1982

[130]Marc B. Vilain, Henry A. Kautz - *Constraint Propagation Algorithms for Temporal Reasoning*. Proceedings a 5-a Conferință AAAI Conference, pag. 377-382, Philadelphia, SUA, 1986

[131]Thang Vu Dong - *Découverte de chronique à partir de journaux d'alarmes : application à la supervision de réseau de télécommunication*, Capitolul 2: *Répresentation,* Teză de doctorat nr. 1772, Institut National Polytechnique de Toulouse, Franța, 2001

[132]Eric W. Weisstein - *Hasse Diagram*, http://mathworld.wolfram.com/HasseDiagram.html

[133]http://dexonline.ro – site web DEX online, dicționar explicativ al limbii române

[134]http://crs.elibel.tm.fr/en/index.html - site web pentru sistemul CRS (Chronicle Recognition System)

[135]http://www.cs.wustl.edu/~schmidt/ACE.html - site web al creatorului interfeței ACE (Adaptive Communication Environment), Dr. Douglas C.Smith, University of California, Irvine, SUA

[136]http://magda.elibel.tm.fr – site web al proiectului MAGDA (Modelisation et Apprentissage pour une Gestion Distribuée des Alarmes)

[137]http://www.omnetpp.org – site web al comunității programatorilor OMNeT++

[138]http://www.omnest.com – site web al versiunii comerciale de OMNeT++

[139]http://www.prismtechnologies.com – site web al proiectului OpenFusion, dezvoltat cu ACE

[140]http://www.riverace.com – site web de suport pentru interfața de comunicații ACE