POLITEHNICA University of Timisoara
Department of Computer Science & Engineering
Digital Signal Processing Laboratories - DSPLabs
2, V. Parvan Bv., 300223 - Timisoara, ROMANIA
Tel: +40 256 403271, Fax: + 40 256 403214
Web: http://dsplabs.utt.ro

Mihai V. MICEA

Proiectarea şi implementarea sistemelor timp-real pentru aplicaţii critice de achiziţie şi prelucrare numerică de semnal

Teză de doctorat

Conducător ştiinţific
Prof. dr. ing. Vladimir CREŢU

2004

Celor două familii ale mele:
Cea de acasă şi cea de la şcoală.

Cuvânt înainte

Mulţumesc domnului prof. dr. ing. Vladimir Creţu, conducătorul ştiinţific de doctorat, pentru ajutorul şi îndrumarea continuă şi plină de răbdare, acordate nu numai pe parcursul urcuşului lung şi abrupt al activităţii mele de doctorat, ci în toţi anii de excelentă colaborare. Mai mult decât profesor şi coleg, domnul Creţu reprezintă pentru mine un adevărat prieten.

Activitatea mea de cercetare la doctorat a beneficiat de sprijinul preţios al multor colegi din cadrul Universităţii "Politehnica" din Timişoara şi din cadrul Departamentului de Calculatoare. Recunoştinţa mea se îndreaptă către prof. dr. ing. Mircea Stratulat, pentru întregul sprijin moral şi material (echipamente de laborator şi de măsură, documentaţie, etc.). Discuţiile purtate în domeniul sistemelor timp-real cu prof. dr. ing. Nicolae Robu s-au dovedit extrem de utile, fiind germeni ai unor importante idei din lucrare. Menţionez de asemenea colaborarea preţioasă şi sprijinul colegilor ing. Lucian Pătcaş, ing. Zsolt Husz şi ing. Alexandru Joni. Lista poate continua pe multe pagini; mulţumesc tuturor colegilor, datorită cărora simt în fiecare zi că aparţin unei mari şi calde familii.

Programul de colaborare dintre Departamentul Calculatoare şi compania Motorola, Inc. (respectiv, Freescale, Inc.), cu Centrul DSP (MDCR) Bucureşti, a reprezentat un pilon important pe care s-a sprijinit şi construit activitatea mea de doctorat. Mulţumesc pe această cale domnilor dr. ing. Andreas Wild, ing. Paul Marino, ing. Daniel Broşteanu şi ing. Costel Ilaş.

Deşi nu e o specialistă de renume în domeniu, contribuţia cea mai importantă la această teză o are soţia mea Claudia, care m-a înconjurat în toţi aceşti ani cu imensa ei dragoste, răbdare şi devotament. Îi mulţumesc din suflet că a dus la capăt această muncă, umăr la umăr cu mine.

Adânca mea recunoştinţă se îndreaptă către părinţii mei, Maria şi Ioan, mai ales pentru că prin sacrificiile şi dragostea lor am devenit ceea ce sunt astăzi.

În fine, dar nu în cele de pe urmă, mulţumesc bunului Dumnezeu pentru că mi-a dat norocul de a mă putea bucura în fiecare clipă de toate lucrurile minunate cu care El mă înconjoară.

Autorul,
Septembrie 2004

Abstract

Embedded systems and digital signal processing (DSP) systems are widely used in today's digital control applications, requiring in most cases real-time behavior of the hardware-software components. Many applications have a critical impact on the environment and/or on the human factor which they interact with. Examples of such applications include: modern flight control systems, fly-by-wire, auto-pilot, automotive control, industrial mechatronics, nuclear plant surveillance, and so on.

There are two essential characteristics a hardware-software platform has to meet in order to provide correct operation results for critical applications: (a) the entire process of system/application development should include the time coordinate, and (b) the system must provide maximum of predictability for the hard real-time tasks.

Although a very large number of projects have been developed in the field of real-time and embedded systems, both in the industry and the academic communities, there still are many important issues to be addressed. Three problems we consider of key importance are: (i) the large number and variety of real-time system development and analysis methodologies, correlated with the lack of their compatibility and standardization, many of them lacking also in providing proper integration of the time coordination; (ii) most current hardware-software architectures are designed and optimized to provide a good *average case* behavior, while the requirements of real-time applications specify a correct behavior of the system even in the *worst case operating conditions*; (iii) the unrestricted use of interrupts and of the associated asynchronous mechanisms and tasks in real-time systems generates severe predictability problems.

This work addresses the "Design and Implementation of Real-Time Systems for Critical Applications of Digital Signal Acquisition and Processing" topic, providing a set of homogenous solutions to the main problems stated above and to some other related issues.

A uniform set of models for real-time signals and tasks has been developed and proved, based on a proper representation of the time. The *ModX* is the resulting model for hard real-time tasks, defining modular, periodic tasks, which are scheduled and executed in a non-preemptive environment. The asynchronous events occurring in the system are processed using polling techniques and proper task scheduling.

The OPEN-HARTS system (Operating Environment for Hard Real-Time Systems) is a new methodology introduced for the development and implementation of hard real-time systems and applications, based on the previously defined set of models for time, signals and tasks. The methodology specifies the interconnection of two sub-systems: INVERTA (Integrated Visual Environment for Real-Time Application Analysis and Development) and HARETICK (Hard Real-Time Compact Kernel).

In order to provide timing guarantee and to maximize the operation predictability of critical and hard real-time applications, special task scheduling techniques must be considered with care. The thesis focuses on studying and

developing non-preemptive scheduling algorithms for hard real-time tasks, and the associated mechanisms to solve the problem of asynchronous event processing within hard real-time systems.

As a case study for the principles and mechanisms introduced and discussed with the OPEN-HARTS methodology, a test version of the HARETICK kernel has been developed. The kernel is implemented on the Motorola DSP56307EVM platform and provides a fully operational environment for applications containing both hard real-time as well as soft (non-) real-time tasks. The applications used to test the HARETICK kernel proved its correct behavior, according to their functional and timing specifications. For example, a particular application demonstrates that the kernel provides the necessary support for generating perfectly periodic output signals which satisfy all their specified parameters (period, impulse length, etc.), thus proving that the entire system can reach the maximum level of predictability – one of the main objectives of our research.

In nature, as well as in the practice of digital control applications, there are many phenomena and sequences of actions that perform without admitting interrupts. On the other hand, many interruptible operations can also be sliced in non-preemptive sub-sequences. This work proves the theoretical and practical validity of the non-preemptive techniques, used in conjunction with a proper and homogenous set of models for time, events and actions. The simulation and experimental results provided by real-time systems based on non-preemptive principles are similar with those provided by classical systems, with respect to the correctness of the operations performed. Nevertheless, when predictability and timing guarantees are needed, the non-preemptive based systems are the only ones able to operate correctly even in the *worst case conditions*.

Cuprins

SECŢIUNEA I. INTRODUCERE

Secţiunea de faţă conţine capitolele de introducere ale lucrării, în care se prezintă tema şi principalele domenii abordate în cadrul tezei.

Capitolul 2 prezintă o scurtă trecere în revistă a principalelor concepte actuale în domeniul sistemelor timp-real de achiziţie şi prelucrare numerică de semnal (APNS) şi de control digital încorporat (embedded systems).

În Capitolul 3 sunt prezentate o serie de probleme curente în domeniu, fiind evidenţiată de asemenea şi importanţa soluţionării acestora pentru ca proiectarea şi implementarea sistemelor şi aplicaţiilor timp-real stricte să se poată efectua în concordanţă cât mai strânsă cu specificaţiile impuse. Preocupările din cadrul cercetărilor noastre de doctorat, prezentate în materialul de faţă, se axează pe introducerea şi studierea unui set de soluţii la aceste probleme.

1 Introducere

Teza de faţă prezintă contribuţiile proprii, rezultate din activitatea noastră de cercetare-dezvoltare cuprinsă în programul de doctorat cu tema "Contribuţii la achiziţia şi prelucrarea numerică în timp-real a semnalelor utilizând sisteme de calcul distribuite", sub coordonarea ştiinţifică a prof. dr. ing. Vladimir CREŢU.

Sistemele de control digital încorporat (embedded systems) şi achiziţia şi prelucrarea numerică a semnalelor (DSP-based systems) sunt două domenii înrudite, de interes major în preocupările actuale ale comunităţii academice şi industriale, fapt dovedit şi de numărul imens de aplicaţii ce integrează astfel de sisteme în aproape toate domeniile activităţii umane. O cerinţă esenţială impusă acestor sisteme şi aplicaţii este operarea în timp-real, cu garantarea respectării termenelor de timp impuse de specificaţiile de proiectare şi de mediu. Un număr foarte mare de aplicaţii au un impact critic asupra mediului înconjurător şi/sau asupra factorului uman cu care interacţionează. Exemple de aplicaţii critice de achiziţie, prelucrare numerică a semnalelor şi control digital încorporat sunt sistemele de control al zborului din avioanele moderne (*avionics*, *fly-by-wire*, *auto-pilot*), sisteme de navigaţie, sistemele de control ale rachetelor, navetelor şi staţiilor spaţiale, echipamentul de supraveghere şi control al automobilelor (*automotive*), linii de fabricaţie automatizată, sistemele de supraveghere şi control al centralelor nucleare, şi multe altele. Eşecul unor astfel de sisteme în îndeplinirea *la termenele specificate* a sarcinilor programate poate avea consecinţe catastrofale asupra mediului înconjurător, ducând chiar la pierderi de vieţi omeneşti.

În prezent există o cantitate covârşitoare de informaţie ce tratează domeniile de mai sus. Majoritatea documentaţiei necesare activităţii noastre de cercetare-dezvoltare a fost colectată începând cu anul 2000, având acces la o serie de resurse importante, recunoscute la nivel mondial în comunitatea academică şi industrială: societăţile ştiinţifice IEEE şi ACM (SUA), site-uri de Web ale unor instituţii şi laboratoare de renume în domeniu, baze de informaţie ale unor firme majore (Motorola, Texas Instruments, QNX), etc. Informaţia colectată a fost procesată în cea mai mare parte a sa, până în anul 2002. Din materialul studiat, a rezultat faptul că *problematica sistemelor timp-real (TR) stricte destinate aplicaţiilor critice de achiziţie şi prelucrare numerică a semnalelor (APNS) şi de control digital încorporat* nu este rezolvată în mod satisfăcător, existând încă o serie de probleme importante:

- Introducerea timpului ca o coordonată esenţială în toate etapele dezvoltării, analizei şi implementării sistemelor şi aplicaţiilor TR;
- Unificarea şi integrarea tuturor modelelor şi fazelor de dezvoltare ale sistemelor TR (STR) într-o metodologie omogenă;
- Utilizarea cu precauţie a structurilor şi mecanismelor clasice, concepute pentru creşterea eficienţei de operare a sistemelor pentru cazurile de operare tipice (de medie), exemplul cel mai important fiind utilizarea fără restricţii a întreruperilor şi a mecanismelor asincrone;

- Existenţa unei mari varietăţi de metode de specificare şi verificare formală a STR. Până în prezent însă, nici una nu este încă acceptată în întregime de mediul academic şi în industrie. Principala problemă constă din decalajul important între rezultatele cercetărilor din domeniul metodelor formale timp-real (modelele concepute) şi implementările practice reuşite.

Activitatea noastră de cercetare, sintetizată în lucrarea de faţă, îşi propune să aducă o serie de contribuţii şi soluţii pentru rezolvarea problemelor de mai sus. Abordarea din lucrare se axează pe următoarele idei principale:

- Utilizarea întreruperilor în sistemele TR stricte reprezintă o problemă din punctul de vedere al predictibilităţii, afectând capacitatea acestora de a garanta în orice condiţii de operare respectarea termenelor impuse task-urilor TR stricte.
- Pentru a oferi predictibilitate maximă sistemelor TR stricte destinate aplicaţiilor critice de achiziţie şi prelucrare numerică a semnalelor şi de control digital încorporat, este necesară abordarea modelelor non-preemptive în ceea ce priveşte definirea semnalelor, a task-urilor şi conceperea algoritmilor de planificare.
- Dezvoltarea viabilă a sistemelor şi aplicaţiilor TR stricte trebuie să aibă la bază, pentru toate fazele sale, metode şi modele omogene, compatibile, în care timpul este o coordonată cheie.
- În procesul de dezvoltare a aplicaţiilor TR stricte, faza de analiză offline, apriori execuţiei sale pe sistemul ţintă, reprezintă o cerinţă obligatorie.

Principalele obiective propuse în cadrul acestui program de cercetare-dezvoltare sunt strâns legate de ideile cheie enumerate anterior:

(1) Conceperea, dezvoltarea şi demonstrarea unui set omogen de modele pentru semnale şi task-uri TR, bazate pe un sistem corespunzător de reprezentare a timpului real.
(2) Conceperea unei metodologii unitare de dezvoltare şi implementare a sistemelor TR pentru aplicaţii critice de achiziţie şi prelucrare numerică de semnal. Elementul central al tuturor fazelor implicate (proiectare, specificare, verificare, analiză, implementare şi testare) este sistemul de modele realizat în cadrul primului obiectiv.
(3) Studiul detaliat al tehnicilor şi metodelor ce permit operarea unui sistem TR strict cu garantarea respectării termenelor de timp impuse de aplicaţiile critice şi maximizarea predictibilităţii. Interesul a fost focalizat pe studiul algoritmilor de planificare non-preemptivă a seturilor de task-uri timp-real şi a mecanismelor ce trebuie dezvoltate pentru rezolvarea elementelor asincrone din operarea sistemelor TR stricte.

Rezultatele obţinute în cadrul activităţii de doctorat sunt sintetizate în teza de faţă. Datorită volumului relativ mare de informaţie prezentată, teza este structurată în secţiuni.

Prima secţiune conţine capitolele de introducere ale lucrării. Astfel, capitolul următor prezintă o scurtă trecere în revistă a conceptelor actuale, ce vor fi abordate

ca subiecte de interes ale lucrării, din domeniul sistemelor TR şi de control digital încorporat.

Cu toate că domeniul sistemelor TR a beneficiat de o atenţie deosebită din partea comunităţii academice şi industriale în ultimele decenii, există încă o serie de probleme nerezolvate, câteva fiind amintite în paragrafele anterioare. În Capitolul 3 sunt evidenţiate şi discutate un număr de subiecte care necesită încă soluţionare şi care vor fi abordate în capitolele următoare ale lucrării.

Secţiunea a doua prezintă contribuţiile aduse la proiectarea, modelarea şi dezvoltarea sistemelor TR stricte pentru aplicaţii critice. În capitolele secţiunii sunt introduse şi studiate o serie de modele care pot fi utilizate în tratarea omogenă a fazelor de concepţie, proiectare, specificare, verificare şi analiză a STR stricte.

Un model temporal este propus şi analizat (Capitolul 4), urmând a se constitui într-un element de bază pentru toate fazele dezvoltării STR stricte. În continuare sunt clasificate şi studiate semnalele cu care interacţionează sistemele TR şi, utilizând modelul temporal sistem, este derivat un set minimal şi complet de parametri temporali ce pot modela semnalele din punct de vedere al specificării formale. Comportarea în timp a task-urilor TR este de asemenea abordată pe baza aceluiaşi model temporal, rezultând un set de parametri şi relaţii, cu ajutorul cărora este propus şi discutat modelul task-ului TR strict, ModX-ul.

Problema planificării seturilor de task-uri TR este abordată în Capitolul 5, din perspectiva modelelor pentru timp, semnale şi task-uri introduse în Capitolul 4, rezultând un studiu detaliat al algoritmilor de planificare non-preemptivă pentru seturi de ModX-uri (task-uri TR stricte, periodice), atât din perspectiva operării algoritmice, cât şi a operării în cadrul STR (operare online). Tot aici sunt discutate în detaliu aspectele legate de modelarea şi planificarea seturilor de task-uri ce necesită o execuţie fixă în cadrul fiecărei perioade a acestora.

Evaluarea performanţelor şi studiul comparativ al algoritmilor de planificare non-preemptivă este prezentată în Capitolul 6. Ca rezultat, se discută selecţia şi modul de combinare a acelor algoritmi care prezintă o comportare optimă pentru clasa de aplicaţii TR critice asupra căreia se axează preocupările din lucrare.

Modelul unui sistem complet, OPEN-HARTS, care unifică toate conceptele introduse şi discutate în această secţiune, este prezentat în Capitolul 7. Sistemul OPEN-HARTS este conceput ca un set de soluţii pentru problemele ridicate de dezvoltarea unitară a unui sistem sau aplicaţii TR stricte, utilizând modelele de timp sistem şi semnal, şi ModX-urile cu planificare şi execuţie non-preemptivă.

Secţiunea a III-a conţine contribuţiile aduse la implementarea şi evaluarea sistemelor timp-real stricte pentru aplicaţii critice. În capitolele secţiunii sunt prezentate detaliile de implementare şi testare ale unui sistem de operare (nucleu) timp-real cu caracteristici care îi permit garantarea respectării termenelor de timp pentru task-urile critice ale aplicaţiilor de achiziţie şi prelucrare numerică de semnal, sau de control încorporat. Astfel, nucleul de operare HARETICK reprezintă un studiu de caz, complet şi consistent, privind implementarea, testarea şi evaluarea principiilor, modelelor teoretice şi algoritmilor discutaţi în cadrul secţiunii anterioare.

Capitolul 8 trece în revistă caracteristicile generale ale nucleului HARETICK, caracteristici ce derivă din faptul că au fost vizate sistemele DSP şi de control digital încorporat (embedded systems). HARETICK a fost conceput ca un set omogen de componente software sistem de bază care asigură operarea de ansamblu a sistemului

conform principiilor TR vizate. Descrierea acestor componente (tipul, rolul, principiul de operare, detalii semnificative de implementare, etc.) este realizată în acelaşi capitol.

Timpul, ca parametru esenţial de operare al sistemului, este reprezentat şi gestionat cu ajutorul unor structuri şi mecanisme dedicate. Capitolul 9 prezintă elementele de implementare şi de sincronizare ale celor trei structuri de gestionare a timpului din HARETICK: timpul sistem absolut, ceasul de timp-real şi timpul de planificare. În continuare sunt detaliate structurile de date şi mecanismele angajate de nucleu pentru reprezentarea şi gestionarea task-urilor TR stricte (ModX-urilor).

Funcţionarea de principiu a sistemului se bazează pe cele două contexte de execuţie complementare – contextul strict şi cel lejer. Descrierea celor două contexte, a modului de operare al task-urilor din cadrul acestora şi a elementelor ce caracterizează stările task-urilor TR stricte pe parcursul existenţei lor în cadrul nucleului, este realizată în Capitolul 10. Componenta principală a HARETICK – executivul timp-real – este de asemenea prezentată în detaliu, subliniindu-se rolul său de bază: asigurarea execuţiei temporizate a ModX-urilor şi rezolvarea comutării corecte a celor două contexte de execuţie. Finalul capitolului este dedicat prezentării planificatorului contextului strict al nucleului.

Capitolul 11 tratează aplicaţiile dezvoltate şi implementate pentru testarea şi evaluarea practică a operării nucleului HARETICK. Rezultatele obţinute cu ajutorul măsurătorilor practice cu echipamente specializate (osciloscop digital, generator de funcţii programabile, etc.) sunt prezentate şi discutate în detaliu.

Secţiunea a IV-a, cu concluziile desprinse pe parcursul tratării subiectelor propuse în lucrare, un studiu comparativ cu alte sisteme din domeniu, rezumatul contribuţiilor, precum şi perspectivele de continuare a cercetării, încheie teza.

Întreaga activitate de cercetare-dezvoltare din cadrul programului de doctorat a fost realizată în Laboratoarele de Prelucrare Numerică a Semnalelor (DSPLabs) ale Departamentului de Ştiinţa şi Ingineria Calculatoarelor, Universitatea "Politehnica" din Timişoara, laboratoare coordonate de autor şi de către prof. dr. ing. Vladimir CREŢU. Diverse elemente şi rezultate au fost obţinute în colaborare cu Laboratorul de Circuite Integrate şi Sisteme Numerice de Achiziţie (coordonator prof. dr. ing. Mircea STRATULAT), Laboratorul Multidisciplinar de Testare a Maşinilor Electrice ("D109", coordonatori: prof. dr. ing. Marius BIRIESCU şi prof. dr. ing. Vladimir CREŢU) şi Laboratorul pentru Studiul Dinamicii Fluidelor (coordonatori: prof. dr. ing. Liviu ANTON şi prof. dr. ing. Alexandru BAYA).

Începând cu anul 2001, programul de doctorat a beneficiat şi de suportul companiei Motorola, Incorporated, constând, printre altele, din oferirea sistemelor de evaluare pentru procesoare numerice de semnal Motorola DSP56303 EVM, DSP56824 EVM, şi DSP56307 EVM care a fost utilizată ca platformă pentru implementarea şi testarea nucleului HARETICK.

2 Sisteme timp-real de achiziţie şi prelucrare numerică de semnal

Pe măsură ce sistemele digitale de calcul şi control sunt integrate în tot mai multe medii şi aplicaţii din toate domeniile de activitate umane, a crescut importanţa şi interesul acordat sistemelor de achiziţie şi prelucrare numerică a semnalelor, a sistemelor de control digital încorporat (*embedded systems*) şi, implicit, a sistemelor software ce operează pe astfel de platforme.

Sistemele de achiziţie şi prelucrare numerică de semnal (*APNS*) şi cele de control digital încorporat sunt în majoritatea cazurilor utilizate în aplicaţii cu cerinţe de operare în timp real şi, în multe cazuri, implementează funcţii critice de procesare.

Capitolul de faţă reprezintă o scurtă trecere în revistă a unor concepte actuale legate de sistemele şi aplicaţiile APNS timp-real, ce vor fi abordate apoi în restul lucrării, cu accent pe cele cu impact critic asupra mediului şi/sau a operatorilor umani.

2.1 Sisteme de achiziţie şi prelucrare numerică de semnal

Datorită numărului imens de aplicaţii ce implică sisteme numerice de procesare, în ultimii ani au apărut noi tipuri de probleme şi domenii de interes în ingineria sistemelor digitale [Baya 99, Micea 00b, Micea 00e, Micea 00f, Micea 01a, Micea 01b, Micea 01c, Cretu 02, Cretu 03]:

- Interfaţarea sistemelor numerice la mediul înconjurător al aplicaţiilor
- Dezvoltarea de algoritmi eficienţi pentru procesarea datelor şi a semnalelor
- Încapsularea echipamentelor digitale de control şi prelucrare a datelor în produse industriale şi de larg consum
- Rezolvarea cerinţelor de operare timp-real
- Specificarea, proiectarea şi implementarea de arhitecturi şi protocoale eficiente pentru comunicaţii digitale.

Domeniul sistemelor de achiziţie şi prelucrare numerică a semnalelor (APNS) se ocupă în principal cu studierea soluţiilor ce tratează primele două puncte de mai sus (vezi Figura 2-1) [Toma 96, Pentek 96, NatInst 99, Micea 00c].

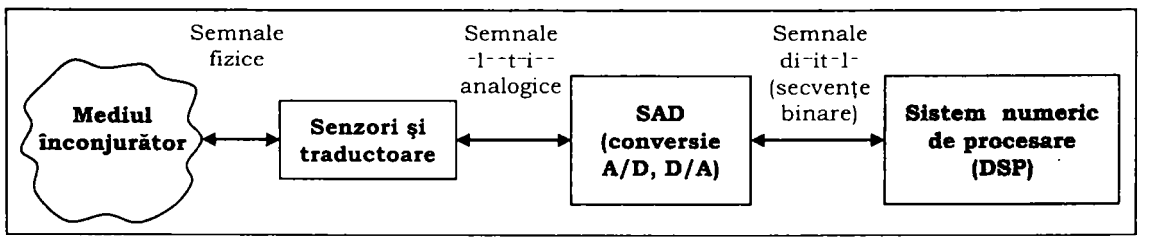

Figura 2-1. Interfaţarea sistemelor digitale la mediul înconjurător

2.1.1 Sisteme de achiziţii de date

Sistemele de achiziţii de date (SAD) sunt dispozitive electronice hibride (analogice şi digitale), cu rolul principal de interfaţă între sistemele digitale de procesare şi mediul înconjurător. Funcţiile de bază ale SAD cuprind următoarele: (a) condiţionarea semnalelor, (b) conversia analog-digitală, (c) conversia digital-analogică, (d) interfaţă digitală cu mediul, şi (e) comunicaţia de date cu sistemul gazdă (cu DSP).

(a) *Condiţionarea semnalelor*

Semnalele fizice provenite (prelevate) din mediul înconjurător (de exemplu: sunet, lumină, mişcare, presiune, umiditate, debit al fluidelor, semnale biologice, unde seismice, etc.) sunt convertite în semnale electrice (curent sau, de regulă, tensiune) cu ajutorul senzorilor şi traductoarelor. Operaţia reciprocă, prin care sistemele digitale (DSP) acţionează asupra mediului, este efectuată cu ajutorul actuatorilor.

Condiţionarea (adaptarea) semnalelor implică operaţii de adaptare a domeniului de scală utilizat de senzori/traductoare şi actuatori la cel utilizat de circuistica de conversie analog-digitală şi digital-analogică din cadrul SAD. Tot la acest nivel se pot aplica filtrări preliminare ale semnalelor, eliminându-se astfel benzile de frecvenţă care nu sunt necesare.

(b) *Conversia analog-digitală*

Conversia analog-digitală (ADC, Analog-to-Digital Conversion) a semnalelor reprezintă funcţia cheie a SAD şi constă din setul de operaţii care stabilesc o corespondenţă directă între o valoare electrică analogică de intrare (curent sau tensiune electrică) şi un cod binar de ieşire, de lungime (precizie) finită.

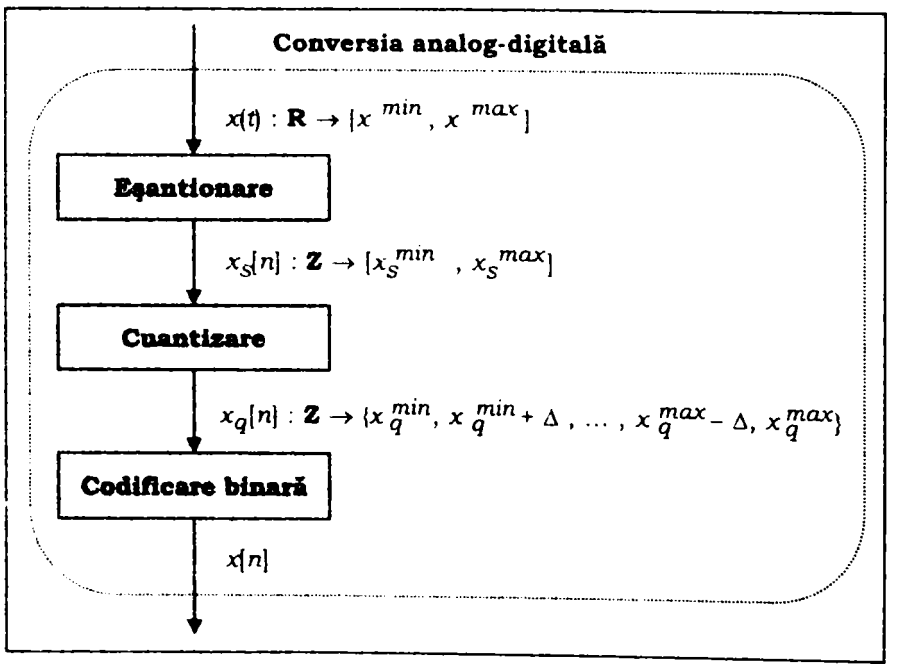

Figura 2-2. Procedura de conversie analog-digitală

ADC este un proces ce se desfăşoară în trei faze (vezi Figura 2-2), executate secvenţial de către un dispozitiv electronic specializat, convertorul analog-numeric [Burr-Brown 95a, Burr-Brown 95b]: eşantionarea, cuantizarea şi codificarea binară.

Eşantionarea reprezintă operaţia de prelevare de eşantioane de valori din semnalul analogic de intrare ($x(t)$ în figura precedentă). De regulă eşantionarea se efectuează la intervale de timp constante (*eşantionare periodică*, cu perioada T_S), rezultând semnalul discret în raport cu timpul $x_S[n]$. Relaţia stabilită de operaţia de eşantionare între cele două variabile de timp, cea continuă t şi cea discretă n, este: $t = n \cdot T_S$, cu $n \in \mathbf{Z}$. Domeniul de valori al semnalului rezultat după eşantionare este tot unul continuu, delimitat de extremele minimă şi maximă ale semnalului de intrare obţinut de la senzori, valorile eşantioanelor fiind astfel exprimate cu precizie infinită (Figura 2-2).

Cuantizarea tratează problema reprezentării cu precizie finită a valorilor eşantioanelor de semnal – problemă comună tuturor sistemelor digitale. Valorile obţinute după cuantizare aparţin unei mulţimi finite, discrete de valori, în care fiecare două elemente consecutive diferă cu valoarea *treptei de cuantizare* Δ, conform relaţiei (2-1):

$$\Delta = \frac{x_q^{\max} - x_q^{\min}}{2^b - 1} = \frac{FSR}{L - 1} \tag{2-1}$$

unde: b este numărul de biţi ce compun codul binar rezultat (rezoluţia conversiei analog-digitale, sau precizia binară a conversiei), *FSR* este domeniul de scală al operaţiei (Full Scale Range), şi $L = 2^b$ reprezintă numărul total de nivele de cuantizare (numărul total de combinaţii binare pe b biţi, sau cardinalul domeniului de valori ale cuantizării).

Codificarea binară poate utiliza mai multe scheme posibile [Albanus 91]: USB (Unipolar Straight Binary), BOB (Bipolar Offset Binary), BTS (Bipolar Two's Complement), etc.

(c) *Conversia digital-analogică*

Conversia digital-analogică (DAC, Digital-to-Analog Conversion) este reciproca procedurii de conversie analog-digitale, punând în corespondenţă fiecare cod binar pe b biţi furnizat la intrare, cu o mărime electrică analogică de ieşire (curent sau tensiune) proporţională [TexasInst 00a, TexasInst 00b].

(d) *Interfaţa digitală cu mediul (Digital I/O)*

În prezent există multe traductoare care transformă mărimile fizice direct în semnale digitale (de exemplu, traductoarele de rotaţie la mouse, traductoare de poziţie, senzori video CCD, etc.). Majoritatea SAD dispun de o interfaţă digitală de intrare-ieşire, paralelă, programabilă şi cu logică tampon.

(e) *Comunicaţia de date cu sistemul gazdă*

Comunicaţia de date dintre SAD şi sistemul gazdă (sistemul de prelucrare a semnalelor digitizate) reprezintă o problemă importantă, ce vizează aspecte ca:

- Viteza sau rata de transfer a datelor: aplicaţiile moderne, ca sisteme multimedia, echipamente de telecomunicaţii digitale, interfeţe evoluate, etc., necesită interfeţe de comunicaţii de înaltă performanţă;
- Arhitectura interfeţelor de comunicaţii: în funcţie de cerinţele aplicaţiilor implementate şi de arhitectura APNS (dacă SAD se găseşte pe o aceeaşi placă sau la distanţă faţă de DSP), interfaţa de comunicaţii va fi de tip local sau la distanţă;
- Complexitatea protocoalelor de comunicaţie;
- Standardizarea: pentru obţinerea unor APNS cât mai modulare, flexibile şi portabile, interfaţa de comunicaţii trebuie să respecte standarde bine specificate. Exemple uzuale: interfaţa serială RS232, USB, FireWire, interfaţa paralelă (SPP, EPP, ECP), CAN, IrDA, PCI, etc.

Arhitectura generală a SAD (vezi Figura 2-3) corespunde principalelor funcţii descrise anterior, fiind compusă din următoarele module [Cretu 99, Micea 00c]: ADCM – modulul de conversie analog-digitală, DACM – modulul de conversie digital-analogică, IOM – interfaţa digitală intrare-ieşire, DAQCDM – modulul de comandă a sistemului de achiziţii, ICOMM – modulul de comunicaţii cu sistemul gazdă, şi IBUS – magistrala internă a SAD (date, adrese şi comenzi).

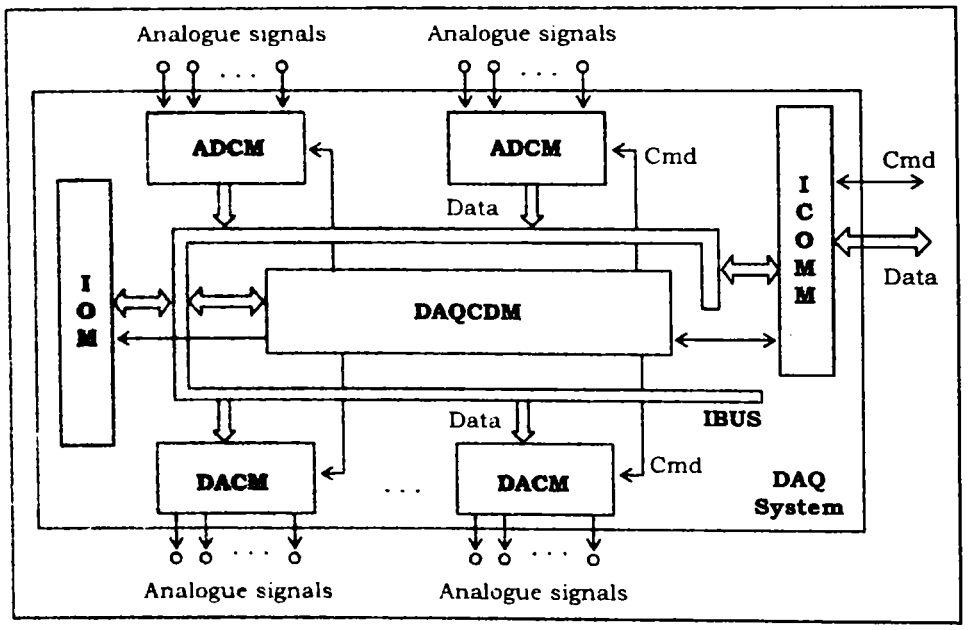

Figura 2-3. Arhitectura generală a SAD

Modulele de conversie analog-digitală (ADCM) şi digital-analogică (DACM) au o arhitectură simetrică (vezi Figura 2-4), incluzând circuite specializate de condiţionare a semnalelor, multiplexare/demultiplexare analogică [TexasInst 00c], eşantionare şi memorare, şi conversie [Burr-Brown 95a]. Semnalele de control

uzuale pentru ADCM şi DACM includ: linii de control ale circuitelor de condiţionare de semnal (de exemplu, controlul coeficientului de amplificare/atenuare a tensiunii de intrare – "Gain Ctrl"), liniile de selecţie a multiplexării, comanda eşantionării şi memorării ("$S/\overline{H}$" – "Sample/Hold"), comanda iniţierii unui nou ciclu de conversie ("SCV" – "Start Conversion"), semnalul ce confirmă terminarea ciclului curent de conversie ("EOC" – "End of Conversion"), etc.

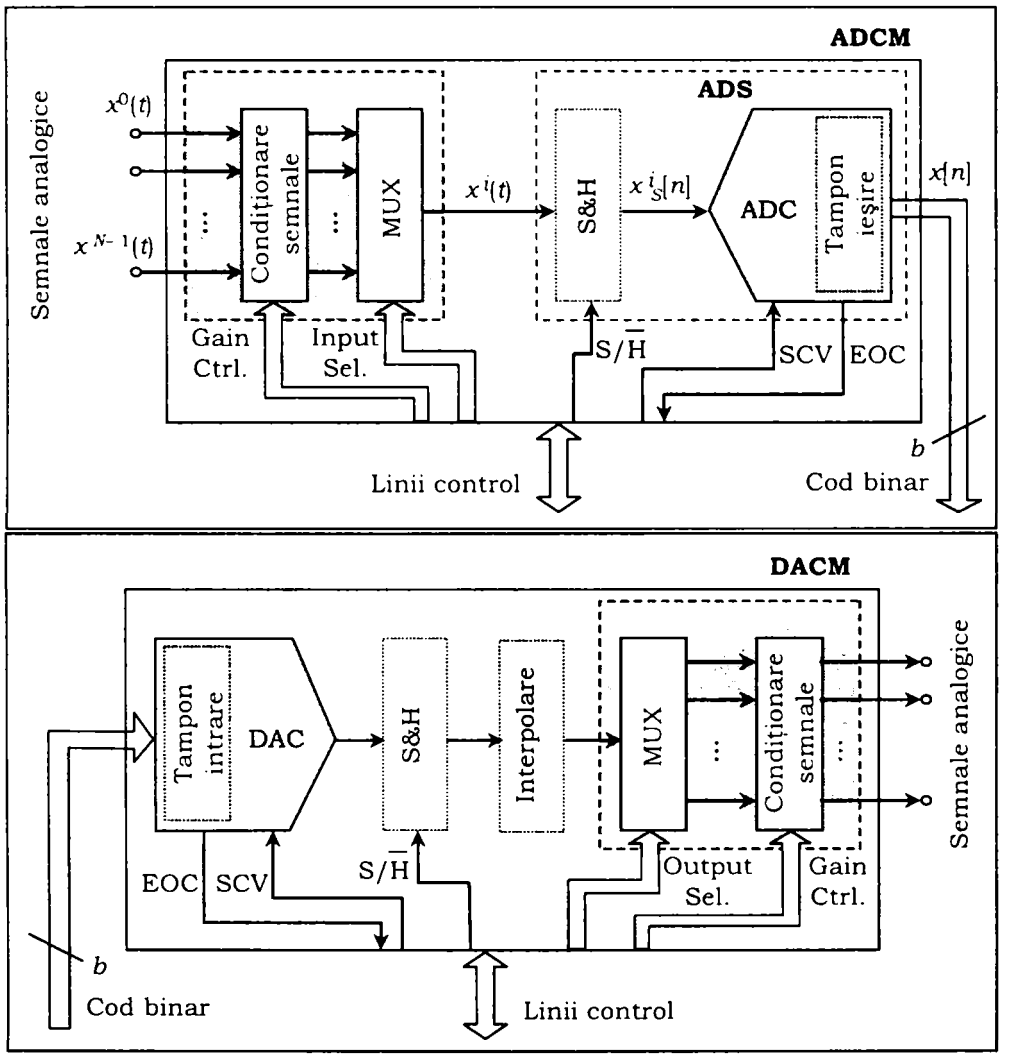

Figura 2-4. Modulul de conversie analog-digitală, ADCM (sus), şi modulul de conversie digital-analogică, DACM (jos)

Controlul operării tuturor modulelor şi blocurilor funcţionale ale SAD este realizat de către modulul de comandă al sistemului (DAQCDM). Un modul tipic de comandă al SAD trebuie să includă următoarele funcţii:

- Decodarea adreselor pentru selecţia diferitelor module ale SAD;
- Implementarea de regiştri programabili speciali de date, comenzi şi de stare, puşi la dispoziţia programelor de aplicaţii de pe sistemul gazdă la care e conectat SAD, pentru controlul cât mai flexibil al achiziţiei;
- Implementarea de mecanisme corespunzătoare de sincronizare a operării componentelor sistemului (de exemplu, pe bază de timere programabile);

- Controlul operaţiilor diferitelor blocuri funcţionale ale SAD: ajustarea amplitudinii semnalelor de intrare sau de ieşire ("Gain Ctrl"), selecţia la multiplexare a canalului analogic curent, extragerea unui eşantion, lansarea unui ciclu de conversie, etc.

Observaţie

Pornind de la tipul de eşantionare (eşantionarea periodică) şi de la modul de operare a dispozitivelor de conversie utilizate în marea majoritate a SAD, semnalele de control a acestor procese sunt strict periodice, având perioade şi termene bine stabilite, care trebuie respectate întocmai pentru ca achiziţia de semnal să se desfăşoare corect.

Aceste cerinţe legate de comanda operaţiilor de achiziţie ale SAD impun o comportare strictă de timp-real pentru sistemele şi aplicaţiile aferente.

2.1.2 Sisteme de prelucrare numerică a semnalelor

Sistemele de prelucrare numerică a semnalelor (DSP – Digital Signal Processing systems) sunt componentele cheie ale oricărui APNS. Un sistem DSP este un dispozitiv electronic, un program sau un algoritm autonom, cu două funcţii principale: (i) prelucrarea semnalelor digitale conform unui algoritm bine definit, şi (ii) controlul şi comanda tranzacţiei de semnale cu mediul înconjurător prin intermediul sistemelor SAD.

Din punct de vedere al implementării sistemelor DSP, există patru abordări curente [Micea 00c]:

(1) *Automate cu stare finită*

Algoritmii de prelucrare numerică necesari unei aplicaţii particulare sunt implementaţi ca un automat cu stare finită, utilizând diverse componente digitale discrete integrate. De exemplu, FPGA (Field Programmable Gate Array) pot fi utilizate pentru implementarea unui astfel de automat.

Avantajele acestei abordări includ viteza mare de procesare şi fiabilitatea înaltă a sistemelor, recomandându-le astfel pentru aplicaţii timp-real critice. Pe de altă parte, există o serie de dezavantaje majore: lipsa acută de flexibilitate şi de scalabilitate, restricţionarea numărului şi a complexităţii algoritmilor de prelucrare numerică ce pot fi implementaţi, limitarea complexităţii generale a sistemului impusă de ingineria şi de tehnologiile curente, etc.

(2) *Sisteme bazate pe microcontrollere*

Microcontrollerele reprezintă în continuare cea mai frecventă soluţie pentru implementarea logicii digitale de procesare şi control în sistemele încorporate, datorită vitezei lor de procesare şi a fiabilităţii relativ mari, combinate cu flexibilitatea oferită de implementarea algoritmilor de procesare ca rutine software.

Arhitectura internă a microcontrollerelor oferă majoritatea resurselor necesare implementării sistemelor de procesare a datelor şi control digital: CPU, memorie

internă RAM şi ROM, porturi intrare-ieşire programabile, timere programabile, mecanisme de întreruperi şi DMA, etc. Cu toate acestea, din cauză că arhitectura microcontrollerelor este destinată procesărilor generice de date, multe aplicaţii DSP ce implică manipulări de date şi calcule complexe la viteze mari nu pot fi implementate corespunzător pe microcontrollere.

(3) *Sisteme bazate pe procesoare specializate (procesoare numerice de semnal)*

Relativ recent a fost dezvoltată o nouă clasă de procesoare specializate, care oferă soluţii eficiente pentru aplicaţiile de prelucrare numerică de semnal. Procesoarele DSP (Digital Signal Processor) derivă din arhitecturile microcontrollerelor, la care au fost aduse îmbunătăţiri în direcţia facilitării execuţiei eficiente a operaţiilor de manipulare a datelor şi a celor aritmetice complexe, implicate frecvent în algoritmii de prelucrare numerică a semnalelor [Motorola 95, TexasInst 97].

Un exemplu de operaţie complexă de prelucrare numerică, implementată cu eficienţă maximă de procesoarele DSP, este *convoluţia* a două semnale digitale [Lynn 92, Ifeachor 93, Motorola 95, Proakis 96, Oppenheim 96]:

$$z[n] = x[n] * y[n] = \sum_k x[k] \cdot y[n-k], \quad k \in \mathbf{Z} \tag{2-2}$$

unde: $x[n]$ şi $y[n]$ sunt cele două semnale digitale procesate, iar $z[n]$ este semnalul rezultat în urma convoluţiei.

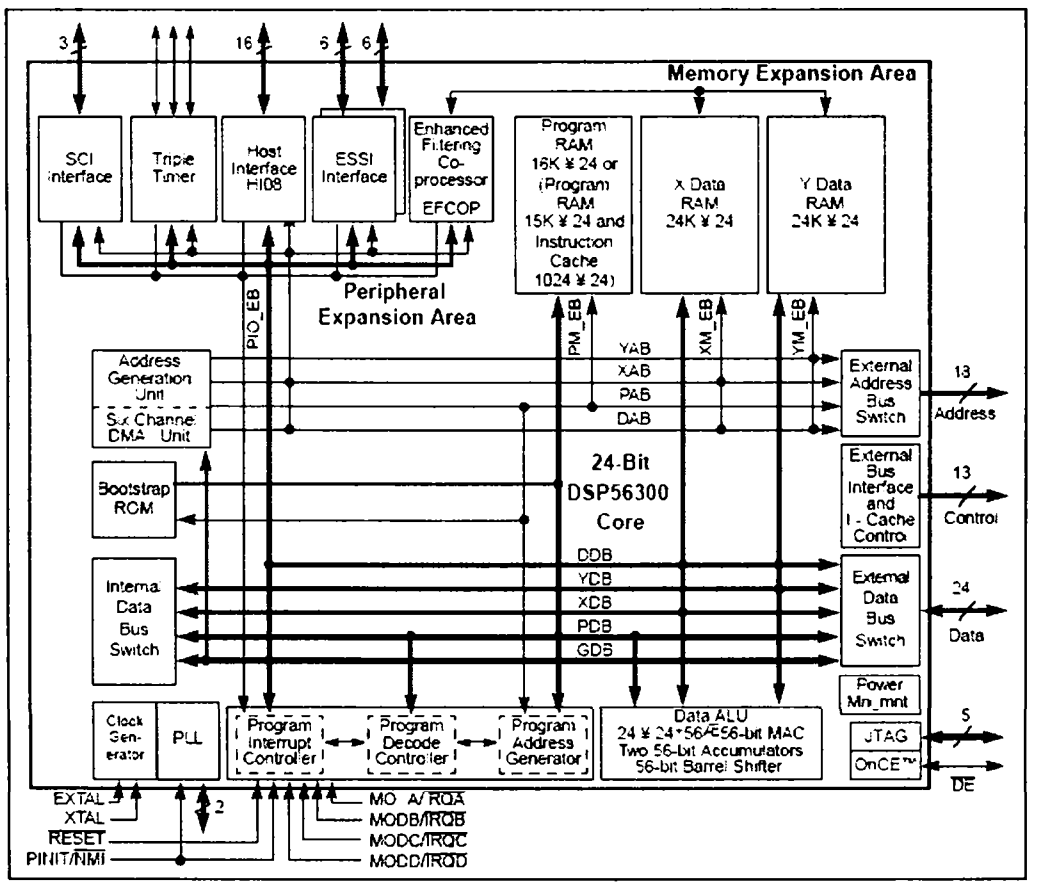

Figura 2-5. Schema bloc a procesorului Motorola DSP56307 (sursa: [Motorola 98])

Arhitectura internă a procesoarelor DSP (exemplificată în Figura 2-5 pentru procesorul Motorola DSP56307 [Motorola 98, Motorola 00]) dispune de blocuri funcţionale care operează autonom în paralel, unităţi aritmetice şi logice care implementează de exemplu operaţia de multiplicare şi acumulare (MAC), necesară în (2-2), într-un singur ciclu maşină, unităţi de generare a adreselor, etc. Arhitectura familiei de procesoare Motorola DSP56k dispune, de exemplu, de următoarele facilităţi suplimentare ce permit implementarea extrem de eficientă a algoritmilor şi programelor de prelucrare numerică de semnal:

- O configuraţie şi interconectare internă a blocurilor funcţionale şi a magistralelor, bazată pe arhitectura Harvard modificată [Motorola 95, Motorola 00], ce permite operarea cu un grad mare de paralelism. De asemenea, interfeţele periferice pot opera, la rândul lor, autonom şi în paralel cu nucleul procesorului;
- Existenţa unui suport puternic pentru diferite tipuri de structuri de date, oferit la nivelul regiştrilor şi a modurilor de adresare: structuri tampon liniare şi circulare, adresare cu deplasament specificat a structurilor tip tabelă, adresare cu inversare de biţi ("bit-reversed addressing") pentru accesarea factorilor implicaţi în calculul Transformatelor Fourier Rapide (FFT), etc.;
- Suport hardware pentru programarea şi execuţia rapidă a buclelor şi a secvenţelor repetitive de instrucţiuni;
- Regiştri ortogonali, ce pot fi utilizaţi de majoritatea instrucţiunilor procesorului ca regiştri de uz general, sau ca regiştri cu funcţii specializate;
- Existenţa a diverse moduri de operare, orientate pe controlul consumului de energie.

Avantajele oferite de procesoarele DSP (performanţă ridicată, costuri relativ scăzute, flexibilitate de programare, consum redus de putere, unelte soft de dezvoltare a aplicaţiilor la costuri moderate, etc.) le recomandă ca una dintre cele mai eficiente soluţii în domeniul prelucrării numerice timp-real a semnalelor şi al controlului digital încorporat [TexasInst 92, Gai 02].

(4) *Software specializat, executat pe calculatoare numerice*

Algoritmii DSP pot fi implementaţi ca programe executate pe calculatoare personale sau specializate. Soluţia are avantajul maximei flexibilităţi şi portabilităţi, prin utilizarea unor limbaje de programare de nivel înalt, ca C sau Java, pentru care există o largă varietate de unelte software de dezvoltare şi analiză.

Pe de altă parte, viteza relativ moderată de execuţie a algoritmilor pe calculatoare ce rulează sub platforme clasice de operare (MS Windows, Linux, etc.), precum şi lipsa de predictibilitate indusă de aceste sisteme de operare, nu recomandă abordarea de faţă ca soluţie pentru sisteme de control autonome, industriale, cu cerinţe stricte de timp-real impuse de multe aplicaţii în domeniu.

2.2 Sisteme timp-real

Oxford Dictionary of Computing dă următoarea definiţie pentru un sistem timp-real [Ostroff 92]:

> "Un sistem timp-real (STR) este orice tip de sistem pentru care timpul de răspuns este un parametru esenţial. Acest lucru se datorează faptului că intrările acestuia corespund unor variaţii din mediul fizic înconjurător în aşa fel încât ieşirile sistemului trebuie să corespundă la rândul lor, aceloraşi variaţii. Decalajul dintre timpii de intrare şi cei de ieşire trebuie să fie suficient de mic, relativ la tipul aplicaţiilor implementate."

[Young 82] defineşte un STR ca fiind: "Orice activitate sau sistem de procesare a informaţiei, care trebuie să răspundă după o perioadă finită şi bine specificată, unor stimuli de intrare generaţi extern."

Aşadar, operarea corectă a STR nu depinde doar de rezultatul logic al procesării, ci şi de timpul în care sunt produse rezultatele [Burns 90, Stankovic 92a, Stankovic 92b, Panzieri 93, Ghosh 94, Parashkevov 94].

Sistemele de calcul pot fi împărţite în trei mari categorii [Berry 00]:

- *Sisteme transformaţionale*: sunt sistemele care procesează un set de date de intrare pentru a obţine un set de date de ieşire, după care se opresc. Majoritatea programelor de procesare numerică se încadrează în această categorie: compilatoare, editoare/procesoare de texte, etc.
- *Sisteme interactive*: interacţionează cu mediul (şi cu operatorii/utilizatorii) într-o manieră de tip "cerere-răspuns" (modelul *master-slave*). Sistemul oferă anumite servicii de procesare, aşteaptă cereri de tip client pe care le deserveşte pe rând. Exemple: sistemele de operare, sistemele de gestionare a bazelor de date, motoare de căutare pe Web, etc.
- *Sisteme reactive*: denumite şi *sisteme reflexe*, reacţionează în mod continuu la semnale de intrare (stimuli) provenind din mediul fizic înconjurător, generând semnale de ieşire (actuatori). Sistemele reactive au o comportare strict legată de cea a intrărilor, trebuind să reacţioneze cu succes în ritmul dictat de mediul înconjurător. Exemple: controloarele de proces, sistemele de control digital încorporat, sistemele APNS, etc.

Sistemele de calcul şi aplicaţiile reale nu aparţin strict doar uneia din categoriile de mai sus. Astfel, sistemele TR sunt, în mod uzual, sisteme reactive, dar care permit un anumit nivel de interacţiune, cu operatorul de exemplu. Figura 2-6 prezintă modelul de principiu al STR, în care sunt evidenţiate sistemul de control (STR), sistemul controlat (mediul înconjurător) şi partea de interacţiune cu operatorul uman. Mediul în care operează un anumit STR are un rol extrem de important în conceperea şi proiectarea acestuia [Stankovic 92b]. O pondere majoră a specificaţiilor STR derivă din restricţiile pentru semnalele de intrare/ieşire impuse sistemului de către mediul înconjurător. De exemplu, perioada de apariţie a unui anumit semnal de intrare are implicaţii directe asupra intervalului maxim de timp pe care STR îl are la dispoziţie pentru a trata acest semnal şi a genera ieşirile corespunzătoare. Acest punct de vedere constituie un element central al preocupărilor noastre legate de proiectarea şi implementarea STR stricte, descrise în lucrarea de faţă.

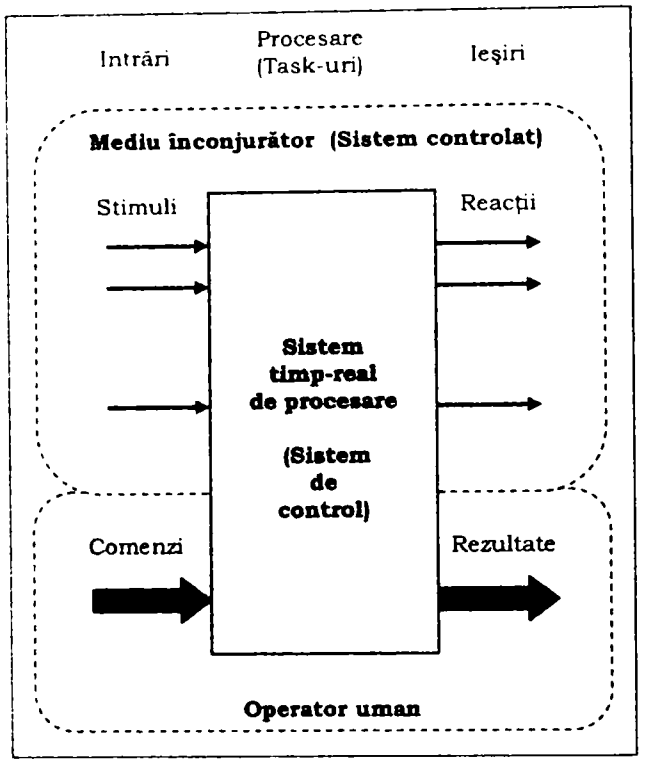

Figura 2-6. Schema de principiu a STR

Multe sisteme controlate (medii înconjurătoare) au caracteristici foarte bine definite (de exemplu, o linie de asamblare, motorul unui automobil, uşi cu operare automată, etc.), fiind clasificate ca *deterministe*. STR care operează în astfel de medii vor fi, de asemenea, deterministe. Un sistem este considerat determinist dacă o aceeaşi secvenţă de intrări va produce de fiecare dată o aceeaşi secvenţă de ieşiri [Berry 00]. Determinismul este o caracteristică de bază a sistemelor reactive şi unul din criteriile care le deosebeşte faţă de sistemele interactive. Identificarea (sau modelarea) unui mediu determinist în care va opera un STR, stă la baza specificării şi proiectării sistemului.

Cerinţele de comportare corectă în timp a STR sunt impuse de impactul fizic pe care îl au sistemele de control asupra mediului înconjurător. Prin urmare timpul devine în cazul sistemelor şi aplicaţiilor TR o coordonată esenţială, cu impact în toate fazele dezvoltării şi exploatării acestora, de la specificare a întregului sistem şi până la execuţia la nivel de task. Cerinţele de timp pentru task-urile STR pot fi clasificate după diverse criterii, cel mai uzual criteriu fiind periodicitatea. De exemplu, majoritatea task-urilor care procesează informaţie obţinută de la senzori sau cele de acţionări asupra mediului sunt de tip periodic şi cu cerinţe de timp severe.

Problema respectării termenelor de timp impuse de aplicaţie (mediu) sistemelor TR le împarte în următoarele categorii:

- *STR critice*: includ funcţii (task-uri) pentru care respectarea termenelor de execuţie este o problemă critică. Exemple de astfel de task-uri sunt cele de monitorizare a temperaturii reactorului într-o centrală nucleară, sistemele de navigaţie ale unei rachete de croazieră sau sistemul tip "fly-by-wire" de control al zborului avioanelor moderne. Nerespectarea termenelor specificate

pentru aceste sisteme are consecinţe catastrofale pentru mediu, pierderi de vieţi omeneşti, etc.

- *STR stricte*: includ task-uri pentru care depăşirea termenelor de execuţie implică o comportare eronată a sistemului. Cu alte cuvinte, valoarea de execuţie a task-urilor TR stricte este anulată în cazul în care execuţia acestora depăşeşte termenele specificate. Nerespectarea termenelor în STR stricte are efecte destul de grave asupra aplicaţiilor ce le utilizează (avarierea sau distrugerea de echipamente şi bunuri materiale, apariţia de erori în ciclul de producţie, anularea de diverse tranzacţii, etc.).
- *STR lejere*: sunt sisteme ce conţin numai funcţii (task-uri) a căror valoare de execuţie nu se anulează, ci se diminuează doar, odată cu depăşirea termenelor specificate. Exemple de astfel de sisteme se găsesc în domeniul multimedia şi al comunicaţiilor uzuale. În cazul unei video-conferinţe, pierderea unui cadru de imagine la recepţie nu este o problemă catastrofală, comunicaţia desfăşurându-se în continuare. Bineînţeles însă că valoarea operării întregului sistem este cu atât mai mare cu cât se pierd mai puţine cadre video.

Tehnicile şi metodele implicate în cazul task-urilor TR stricte (critice) diferă foarte mult faţă de cele utilizate în cazul task-urilor TR lejere. Astfel, pentru STR stricte, este necesară încă din faza de specificare, introducerea unor metode consistente care să permită definirea şi verificarea cât mai corectă a comportamentului în timp. Mai departe, în multe cazuri, task-urile TR stricte sunt pre-alocate (incluzând rezervarea resurselor necesare, pentru intervalele de timp corespunzătoare execuţiei lor) şi pre-planificate, având ca efect garantarea respectării termenelor de execuţie ale acestora. În cazul task-urilor TR lejere, se utilizează frecvent algoritmi de planificare uzuali (care nu sunt de tip TR) sau algoritmi care vizează explicit constrângerile de timp ale task-urilor, dar urmăresc doar performanţa în situaţiile de operare medie (încărcare medie a sistemului, situaţii uzuale de operare, etc.). Aşa cum este evidenţiat şi în [Stankovic 92b], *task-urile TR stricte sunt mult mai dificil de tratat decât cele TR lejere, iar sistemele care operează cu ambele tipuri de task-uri simultan, sunt cu atât mai complexe.*

Fiabilitatea STR reprezintă un subiect extrem de important şi de complex. Problema fiabilităţii trebuie luată în considerare ca un element esenţial încă din faza de proiectare a sistemului TR, implicând atât partea de hardware cât şi partea de software şi trebuie integrată împreună cu constrângerile temporale ale sistemului. Pe lângă tehnicile implicate în garantarea respectării termenelor task-urilor critice şi stricte, în cele mai multe cazuri printr-o analiză "offline" a fezabilităţii şi prin rezervarea apriori a resurselor necesare (chiar dacă acestea vor fi neutilizate o bună parte a timpului de operare), trebuie de asemenea luate în calcul apariţia erorilor şi comportarea sistemului la diferite grade de încărcare.

În concluzie, un sistem TR nu este doar un sistem rapid, ci un sistem *suficient de rapid pentru a putea garanta respectarea termenelor de execuţie a task-urilor, pentru situaţia cea mai defavorabilă de operare.*

2.3 Timpul ca şi coordonată esenţială a specificării şi operării STR

Cercetările din ultimii peste 30 de ani în domeniul sistemelor TR subliniază necesitatea introducerii coordonatei temporale în toate fazele dezvoltării STR: specificare, verificare/validare, programare, analiză, planificare şi execuţie.

Domeniul temporal a fost studiat în numeroase lucrări de specialitate şi au fost propuse diferite modele şi logici temporale [Allen 83, Barbacci 86, Melliar-Smith 87, Jahanian 88, Halpern 91, Ostroff 92, Bucci 95, Mattolini 96, Chen 98, Bellini 00, Berry 00, Mattolini 01]. Aceste modele sunt în general concepute pentru utilizarea în fazele de analiză a cerinţelor şi de specificare a STR, şi se concentrează mai degrabă asupra modelării comportării sistemelor decât asupra aspectelor structurale şi funcţionale ale acestora.

Comportarea unui sistem se referă la modul în care acesta reacţionează la stimulii externi şi la evenimentele interne – aspect critic al sistemelor reactive (şi deci, al STR). Aspectele *structurale* se referă la modul de descompunere a sistemului în subsisteme, iar aspectele *funcţionale* se referă la transformările informaţiei în cadrul sistemului (procesarea informaţiei).

Logicile temporale sunt definite ca structuri de forma:

$$\langle \Im, R, V \rangle \tag{2-3}$$

unde $\Im$ reprezintă domeniul temporal, $R \subseteq \Im \times \Im$ reprezintă relaţia definită de logică asupra elementelor domeniului $\Im$ şi V este funcţia de evaluare a formulelor (propoziţiilor) logicii:

$$V : F \times \Im \rightarrow \{\mathsf{T}, \perp\} \tag{2-4}$$

În (2-4), F este setul de formule ale logicii, iar $\{\mathsf{T}, \perp\}$ – mulţimea valorilor de adevăr ("adevărat" şi "fals"). Funcţia V asignează câte o valoare de adevăr fiecărei formule (propoziţii) ale logicii temporale.

Pentru logicile temporale, relaţia R este denumită *relaţia de precedenţă* şi e notată cu "<". De regulă, "<" este o relaţie tranzitivă şi nereflexivă, realizând o ordonare parţială a domeniului temporal, $\Im$ [Bellini 00]. Proprietăţile pe care relaţia "<" le conferă domeniului temporal sunt următoarele:

Proprietatea 2-1. Tranzitivitatea
$(t_i < t_j) \wedge (t_j < t_k) \Rightarrow (t_i < t_k)$, pentru $\forall\, t_i, t_j, t_k \in \Im$.

Proprietatea 2-2. Nereflexivitatea
$-\, t < t$, pentru $\forall\, t \in \Im^+$.

Proprietatea 2-3. Liniaritatea
$(t_i < t_j) \vee (t_i = t_j) \vee (t_j < t_i)$, pentru $\forall\, t_i, t_j \in \Im$.

Proprietatea 2-4. Predecesorul
Pentru $\forall\, t_i \in \Im$, $\exists\, t_j \in \Im$, astfel încât $t_j < t_i$.

Proprietatea 2-5. Succesorul
Pentru $\forall\ t_i \in \Im, \exists\ t_j \in \Im,$ astfel încât $t_i < t_j$.

Proprietatea 2-6. Limita stângă (începutul)
$\exists\ t_i \in \Im$, pentru care $\neg \exists\ t_j \in \Im,$ astfel încât $t_j < t_i$.

Proprietatea 2-7. Limita dreaptă (sfârşitul)
$\exists\ t_i \in \Im$, pentru care $\neg \exists\ t_j \in \Im,$ astfel încât $t_i < t_j$.

Proprietatea 2-8. Domeniul continuu
Pentru $\forall\ t_i, t_j \in \Im$ cu $t_i < t_j \Rightarrow \exists\ t_k \in \Im,$ astfel încât $t_i < t_k < t_j$.

Proprietatea 2-9. Domeniul discret
Pentru $\forall\ t_i \in \Im,\ \exists\ t_j \in \Im,$ cu $t_i < t_j,$ pentru care $\neg \exists\ t_k \in \Im,$
astfel încât $t_i < t_k < t_j$.

Proprietatea 2-6 exprimă faptul că domeniul temporal este limitat în trecut, iar Proprietatea 2-7, că e limitat în viitor. Proprietatea 2-8 semnifică faptul că între oricare două instanţe de timp există întotdeauna o a treia. Se observă că există proprietăţi care se exclud reciproc, cum sunt de exemplu, Proprietatea 2-8 cu Proprietatea 2-9. Fiecare din aceste proprietăţi definesc câte o structură particulară pentru domeniul temporal utilizat. Astfel, de exemplu, sistemele digitale de calcul utilizează un domeniu temporal liniar, discret şi limitat în trecut (la momentul $t_0 = 0$). În acest fel, se poate asocia câte o stare a sistemului pentru fiecare instanţă de timp. Pe de altă parte, un sistem fizic oarecare poate utiliza un domeniu temporal liniar, continuu şi nelimitat.

Pe o structură particulară a domeniului temporal, timpul poate fi modelat după două metode principale: logici punctuale şi logici bazate pe intervale.

Logicile temporale punctuale se bazează pe exprimarea relaţiilor dintre evenimente în termeni de instanţe de timp (puncte din domeniul temporal). Aceste logici definesc intervalele ca seturi conectate de puncte succesive. Exprimarea relaţiilor dintre intervalele în care se verifică anumite evenimente este mai dificilă utilizând modelarea punctuală a timpului.

Logicile temporale bazate pe intervale sunt mult mai expresive, fiind capabile a descrie evenimente în cadrul unui interval de timp. O instanţă de timp (un punct din domeniul temporal) este reprezentată ca un interval de dimensiune (durată) unitară. Relaţiile dintre intervale sunt exprimate mai uşor şi mai intuitiv în aceste modele, utilizând operatori specifici comparativi (exprimă relaţiile dintre intervale) şi combinatorici (exprimă modul de combinare al intervalelor), sau operatori care precizează limitele intervalelor pe baza valorilor de adevăr ale predicatelor. Figura 2-7 ilustrează relaţiile posibile între două intervale, împreună cu operatorii comparativi aferenţi, conform modelului temporal bazat pe intervale propus în [Allen 83] şi [Allen 94].

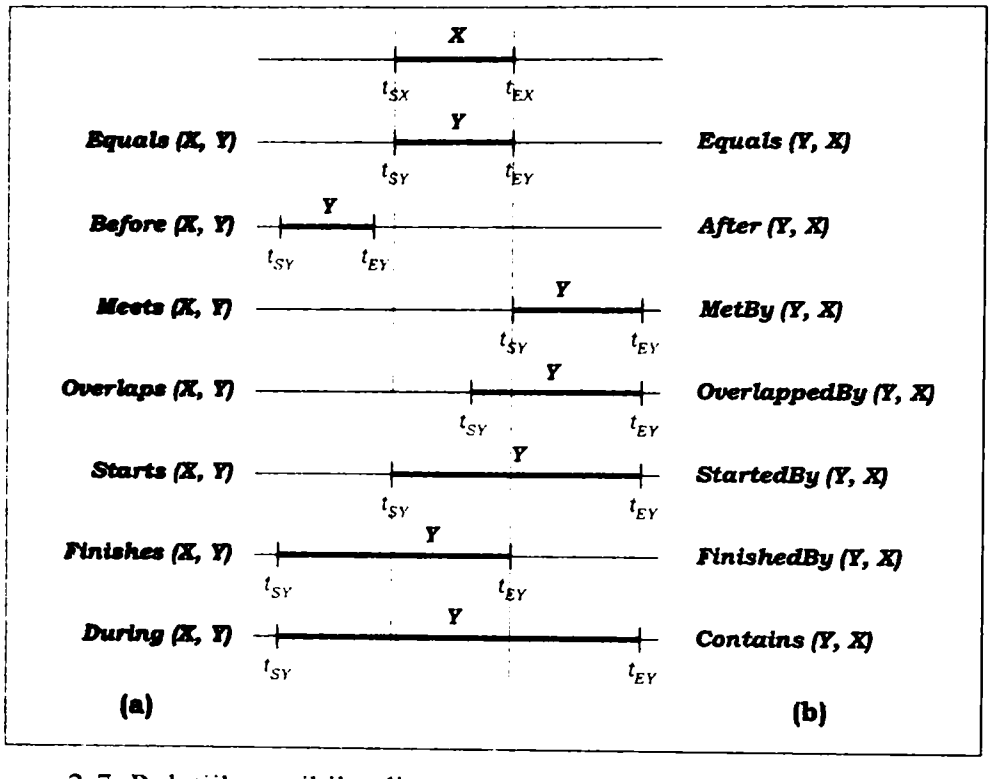

Figura 2-7. Relaţiile posibile, directe (a) şi reciproce (b), între două intervale temporale

Operatorii utilizaţi în modelul temporal ilustrat în Figura 2-7 pot fi transformaţi în operatori ce precizează limitele intervalelor, prin adăugarea unei *metrici* pentru timp şi utilizând operatorul elementar □ ("totdeauna, în viitor", ce exprimă conceptul de necesitate în viitor). Astfel,

$$\Box_{[2,9]} A \tag{2-5}$$

semnifică faptul că formula (propoziţia) A este adevărată pentru toate instanţele de timp, de la cea de-a doua şi până la cea de-a noua, de la momentul curent de timp în care se face evaluarea. În aceste condiţii, operatorul *Before*(X,Y) din Figura 2-7 va fi echivalent cu:

$$\Box_{[tsx,\ tex]} X \Rightarrow (\Box_{[tsy,\ tey]}\ Y) \wedge (t_{ey} < t_{sx})$$

Logicile temporale cu metrică de timp permit definirea de relaţii temporale cantitative, cum ar fi distanţa dintre două evenimente sau durata unor evenimente, exprimate în unităţi de timp. *Posibilitatea exprimării de constrângeri temporale cantitative este esenţială pentru specificarea STR.*

Timpul poate fi modelat în manieră *implicită* sau *explicită*, şi *relativă* sau *absolută*. Un model temporal este implicit atunci când semnificaţia formulelor depinde de evaluarea momentului curent al timpului, acesta fiind considerat implicit în formule. De exemplu, formula (2-5) este echivalentă cu:

$$\forall t \in [t_0 + 2, t_0 + 9].A(t)$$

unde t_0 este evaluarea timpului (instanţa curentă de timp). Pe de altă parte, modelul temporal explicit exprimă timpul curent prin intermediul unei variabile, prezentă în toate formulele modelului. Specificarea explicită a timpului permite exprimarea

oricărei proprietăţi utile a timpului-real. De asemenea, modelele temporale explicite permit specificarea de expresii care nu au sens în domeniul temporal, cum ar fi, de exemplu, activarea unui predicat pentru instanţele pare de timp [Bellini 00].

Referirea timpului este considerată absolută când valoarea timpului curent este determinată în funcţie de un ceas sistem general, idealizat (se presupune că nu există nici un fel de devieri ale ceasului de la timpul absolut). În aceste modele, duratele intervalelor şi momentele instanţelor de timp sunt exprimate direct în secunde sau milisecunde, etc. (adică au valori absolute, corespunzătoare ceasului referit). Modelele temporale relative exprimă duratele şi instanţele de timp în unităţi temporale, relativ la o instanţă a timpului absolut, exprimată în secunde, milisecunde, etc. Determinarea valorii absolute a elementelor temporale pentru modelul relativ, este amânată pentru faza implementării sistemului.

2.4 Tehnici de planificare

Rolul algoritmilor de planificare în cadrul STR este găsirea de soluţii fiabile pentru asignarea în timp a resursei principale a sistemului – procesorul (sau procesoarele), către task-urile sistemului, astfel încât să nu existe suprapuneri ale execuţiilor acestora pe parcursul operării.

În literatura curentă există studii teoretice, simulări şi implementări efectuate asupra unui număr extrem de mare de algoritmi de planificare pentru STR [Liu 73, Kim 80, Jeffay 91, Mercer 92, Panzieri 93a, Panzieri 93b, Ghosh 94, Ramamritham 94, Howell 95, George 96, Kang 98, Letia 00, Radulescu 02]. La proiectarea unui STR, alegerea unui anumit algoritm de planificare (sau a unei politici de planificare) poate depinde de o varietate de considerente, ca: numărul de procesoare ale sistemului, omogenitatea sau eterogenitatea acestora, modul lor de cuplare, tipul aplicaţiilor ce vor fi executate în sistem, etc. Aşa cum se subliniază şi în [Ramamritham 94], date fiind numărul şi diversitatea extrem de mari a abordărilor legate de algoritmii de planificare, o analiză exhaustivă a domeniului ar fi imposibilă. Chiar şi o clasificare unitară şi sintetică a tehnicilor de planificare este dificil de realizat. Câteva criterii de clasificare a algoritmilor de planificare pentru STR sunt trecute în revistă în continuare:

1) După numărul procesoarelor sistemului:

- algoritmi mono-procesor;
- algoritmi multiprocesor.

2) După modul de cuplare a procesoarelor sistemului:

- algoritmi pentru sisteme multiprocesor (cu memorie partajată);
- algoritmi pentru sisteme distribuite.

3) După modul (momentul) de generare a planificării:

- algoritmi statici (planificare generată offline);
- algoritmi dinamici (planificare generată online, în timpul operării sistemului).

4) În funcţie de acceptarea sau nu a întreruperilor:

- algoritmi preemptivi (se admite întreruperea task-ului planificat şi executat curent în sistem, de către un altul cu prioritate mai mare, de exemplu);
- algoritmi non-preemptivi (pe durata execuţiei unui task nu sunt admise întreruperi).

Un factor deosebit de important ce influenţează politicile de planificare pentru STR îl reprezintă necesarul de resurse sistem (cuantificate mai ales în timp de procesare) pe care îl implică execuţia algoritmilor de planificare. Importanţa acestui criteriu derivă şi din considerentul că, practic, task-ul de planificare nu aparţine aplicaţiei propriu-zise; utilizarea procesor corespunzătoare task-ului de planificare reducând astfel, resursele puse de sistem la dispoziţia aplicaţiei. Există două caracteristici care afectează puternic necesarul de resurse sistem, impus de către algoritmii de planificare: complexitatea algoritmilor şi posibilitatea efectuării unei analize offline a planificabilităţii aplicaţiei.

Algoritmii de planificare statică, mono-procesor, au complexitatea cea mai mică, fiind astfel studiaţi şi implementaţi în majoritatea sistemelor TR ce permit această abordare. La capătul opus al scalei complexităţii, se situează algoritmii de planificare dinamică, destinaţi sistemelor multiprocesor şi distribuite.

Algoritmii de planificare statică presupun existenţa unei analize offline a planificabilităţii aplicaţiei ce urmează a fi executată pe STR. Rezultatul analizei constă într-o tabelă care specifică planificarea şi execuţia fiecărui task în parte, împreună cu timpii lor de execuţie, sau este stabilită (prin asignarea de priorităţi, de exemplu) ordinea în care executivul STR va lansa în execuţie task-urile aplicaţiei. Cei mai frecvenţi algoritmi de planificare statică utilizaţi în cadrul sistemelor TR sunt algoritmii de *planificare ciclică* [Ramamritham 94, Ghosh 94] şi algoritmul *RMS* (*Rate Monotonic Scheduler*) [Liu 73, Kim 80, Audsley 91]. De exemplu, RMS asignează priorităţi distincte pentru fiecare task al aplicaţiei, în manieră monoton descrescătoare a frecvenţei de execuţie a acestora, astfel încât prioritatea mai mare este atribuită task-ului cu frecvenţa mai mare şi aşa mai departe.

Avantajul tehnicilor statice de planificare constă în obţinerea unei predictibilităţi mari a operării sistemului, cât şi în complexitatea redusă a algoritmilor, facilitând astfel implementarea acestora pe arhitecturi simple, destinate unor aplicaţii bine precizate, de control digital al mediilor complet deterministe. Există însă o serie de dezavantaje majore: lipsa acută de flexibilitate (modificarea unui parametru al unui task oarecare, sau modificare unei condiţii de operare sau de mediu, implică reluarea analizei offline şi recalcularea întregii planificări), utilizarea procesor relativ mică a aplicaţiilor pe care algoritmii sunt capabili a le planifica şi fenomenul de inversare a priorităţilor (*priority inversion* – un task cu prioritate mai mare, ce doreşte să acceseze o anumită resursă partajată a sistemului, este blocat de către un task cu

prioritate mai mică, ce a fost întrerupt de către primul în timpul accesării aceleiaşi resurse).

Dezavantajele algoritmilor statici de planificare enumerate mai sus au constituit unul dintre motivele principale pentru care au fost concepuţi şi studiaţi algoritmii de planificare dinamică. Aceşti algoritmi asignează în mod dinamic priorităţile task-urilor aplicaţiei, în timpul operării sistemului, aplicând diverse criterii pentru selectarea task-ului ce urmează a fi programat şi executat la momentul curent. *EDF* (*Earliest Deadline First*), *MLF* (*Minimum Laxity First*) şi *MUF* (*Maximum Urgency First*) [Liu 73, Ghosh 94, Ramamritham 94, Panzieri 93a, Panzieri 93b] sunt trei dintre cei mai eficienţi algoritmi de planificare dinamică, deşi s-a demonstrat că, în contextele preemptive de operare, nu se poate garanta respectarea termenelor de timp ale task-urilor, în situaţii tranzitorii de supraîncărcare a sistemului.

EDF asignează în mod dinamic priorităţi task-urilor aplicaţiei după criteriul celui mai apropiat termen de execuţie (*deadline*): task-ul cu cel mai apropiat termen primeşte prioritatea cea mai mare. *MLF*, reprezintă o versiune modificată a lui *EDF*, în care se ţine cont şi de durata de execuţie a task-urilor, nu numai de termenele de execuţie. *Intervalul disponibil planificării* (*laxity interval*) este un parametru ce caracterizează comportarea temporală a task-urilor STR, fiind definit ca intervalul de timp rămas, din momentul curent, până la expirarea termenului de execuţie, considerând durata maximă de execuţie a task-ului. Intuitiv, laxitatea măsoară flexibilitatea (lejeritatea) disponibilă la un moment dat pentru planificarea unui task. *MLF* asignează prioritatea cea mai mare task-ului cu laxitatea cea mai mică. O altă variantă a lui *EDF*, care încearcă să îmbunătăţească predictibilitatea mai ales în situaţiile tranzitorii de supraîncărcare a sistemului, este *MUF*. Algoritmul prevede pentru fiecare task, asignarea (statică) a unui parametru ce specifică în mod explicit gradul de urgenţă, şi determinarea dinamică a unei priorităţi care este invers proporţională cu laxitatea task-ului (ca în cazul *MLF*). Gradul de urgenţă al task-urilor este o combinaţie a două priorităţi fixe: (a) nivelul critic, cu precedenţă mai mare decât prioritatea dinamică a task-ului, şi (b) prioritatea utilizator, cu precedenţă mai mică decât prioritatea dinamică. Prin această combinaţie, se încearcă ajutarea algoritmilor de planificare în diferenţierea dintre task-uri mai importante şi mai puţin importante, prin noţiunea de "prioritate" specificată de utilizator.

Puternica răspândire a sistemelor de calcul distribuite din ultimul deceniu a impus necesitatea abordării sistemelor TR distribuite, în cadrul căruia, planificarea reprezintă un subiect central de interes. Numărul foarte mare de tehnici de planificare propuse şi studiate dovedeşte atenţia de care se bucură domeniul STR distribuite în prezent. Planificarea task-urilor în STR distribuite se află la capătul superior al scalei, atât din punct de vedere al complexităţii algoritmilor (de ordin nepolinomial, NP [Rădulescu 02]), cât şi al resurselor de calcul (timp) necesare. În consecinţă, planificarea task-urilor într-un sistem distribuit nu mai poate aplica seturi de criterii bine definite, în operaţii de căutare exthaustivă pentru selecţia task-urilor ce vor fi planificate, ci sunt necesare abordări euristice, probabilistice şi de aproximare.

Au fost propuse diverse metode pentru măsurarea şi compararea performanţelor algoritmilor de planificare. *Raportul de utilizare procesor* (sau simplu, *utilizarea procesor – utilization factor*), discutat în detaliu în [Liu 73], reprezintă un parametru de bază, utilizat atât ca şi criteriu de apreciere a performanţei algoritmilor de

planificare, cât şi pentru derivarea condiţiilor de fezabilitate a planificării (vezi şi Capitolul 5):

$$U^{\boldsymbol{M}} = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{2-6}$$

unde: C_i reprezintă durata de execuţie a task-ului i din setul $\boldsymbol{M}$ de task-uri supus planificării, T_i este perioada task-ului i, şi $n = |\boldsymbol{M}|$ este numărul de task-uri din $\boldsymbol{M}$.

Pe baza factorului de utilizare procesor, se defineşte parametrul denumit *limita superioară a planificabilităţii* (*least upper bound to the processor utilization factor* [Liu 73], *SB – schedulable bound* [Ghosh 94], *BU – breakdown utilization* [Panzieri 93a, Panzieri 93b]), reprezentând utilizarea procesor maximă pentru care se poate garanta respectarea termenelor impuse task-urilor dintr-un set supus algoritmului de planificare evaluat:

$$SB_A (= BU_A) = \max_{\forall \boldsymbol{M}} \left\{ U^{\boldsymbol{M}} \mid \boldsymbol{M} \text{ e fezabil cu } A \right\} \tag{2-7}$$

unde A este algoritmul de planificare evaluat.

Evaluarea algoritmilor de planificare cu ajutorul limitei superioare a planificabilităţii exprimată de (2-7), dă următoarele rezultate, spre exemplu pentru algoritmul *RMS* [Ghosh 94]: $SB_{RMS} = 0.693$ pentru seturi generice de task-uri (de dimensiuni apropiate de infinit), $SB_{RMS} = 0.88$ pentru cazurile în care perioadele task-urilor au o distribuţie uniformă, şi $SB_{RMS} = 1.00$ doar în cazul în care perioadele task-urilor sunt armonice ale celei mai mici perioade din set. Pe de altă parte, algoritmul *EDF* se caracterizează printr-o limită superioară a planificabilităţii unitară (100%) pentru toate seturile generice de task-uri ($SB_{EDF} = 1.00$), dar, aşa cum a fost menţionat anterior, algoritmul nu poate garanta respectarea termenelor tuturor task-urilor pentru cazurile de supraîncărcare tranzitorie a sistemului.

Alte exemple de parametri destinaţi evaluării performanţelor algoritmilor de planificare sunt [Panzieri 93a, Panzieri 93b]: *timpul mediu, normalizat, de răspuns NMRT* (*Normalized Mean Response Time*) şi *factorul de garantare GR* (*Guaranteed Ratio*). *NMRT* este definit ca raportul dintre intervalul de timp scurs din momentul în care un task este gata de execuţie şi până se termină execuţia lui, şi timpul procesor consumat pentru execuţia efectivă a task-ului. Cu cât *NMRT* este mai mare, cu atât este mai mare timpul "de inactivitate" al task-urilor, eficienţa algoritmului fiind deci mai scăzută.

Factorul de garantare *GU*, reprezintă raportul dintre numărul de task-uri pentru care algoritmul evaluat poate garanta execuţia cu respectarea termenelor, faţă de totalul task-urilor care sunt gata de execuţie.

Din punctul de vedere al sistemelor TR stricte, destinate aplicaţiilor critice, predictibilitatea operării este o cerinţă esenţială. Deşi s-a demonstrat faptul că algoritmii de planificare non-preemptivă necesită – în cazul general – resurse (timpi) de calcul cu ordin de complexitate strict nepolinomial (NP-hard) [Cheng 88, Jeffay 91, Panzieri 93a], ei prezintă potenţialul garantării execuţiei task-urilor cu respectarea termenelor specificate, chiar şi în condiţiile cele mai dezavantajoase de operare ale sistemului (*worst case operation*), spre deosebire de abordările preemptive.

Având în vedere acest considerent (vezi şi Capitolul 3), în lucrarea de faţă vom aborda problema planificării non-preemptive a seturilor de task-uri modelate special pentru sisteme TR stricte.

2.5 Exemple de sisteme timp-real

În prezent există, în diverse faze de realizare (de la specificare şi modelare, până la variante şi prototipuri funcţionale), o varietate extrem de mare de sisteme timp-real şi metodologii de dezvoltare a aplicaţiilor TR. Nici unul dintre acestea însă nu a ajuns să se impună în mediul academic şi industrial în aşa fel încât să se poată vorbi de o standardizare în domeniu. Câteva motive pentru această situaţie vor fi discutate în Capitolul 3.

În paragrafele următoare se vor trece pe scurt în revistă câteva sisteme TR şi metodologii de dezvoltare a acestora, pe care le-am considerat semnificative din perspectiva stadiului de realizare a acestora (a funcţionalităţii), precum şi a tipurilor de probleme timp-real pe care le tratează.

2.5.1 Sistemul Giotto

Sistemul "Giotto" reprezintă o metodologie completă de programare pentru sisteme de control digital încorporat, mono-procesor sau distribuite, cu capabilităţi timp-real stricte [Henzinger 03b], dezvoltată la "University of California at Berkeley", Statele Unite.

Sistemul "Giotto" se bazează pe două componente principale: (1) limbajul de programare şi compilatorul Giotto şi (2) mediul (micro-nucleul) de operare ("run-time system", "programmable microkernel") pentru sisteme TR.

(1) *Limbajul de programare şi compilatorul Giotto*

Limbajul de programare Giotto [Henzinger 01, Henzinger 03a], defineşte arhitecturi software în cadrul cărora se poate specifica şi programa comportamentul funcţional şi temporal al aplicaţiilor timp-real. Limbajul Giotto se bazează pe următoarele paradigme legate de STR: evenimentele şi acţiunile sunt periodice şi sincronizate, iar operarea STR este temporizată ("time-triggered systems").

Din punct de vedere formal, limbajul Giotto prezintă următoarele componente de bază:

(a) *Task-urile*: reprezintă elementele funcţionale ale unei aplicaţii. Task-urile Giotto sunt periodice şi modulare.

(b) *Modurile*: reprezintă o grupare a unei anumite configuraţii de invocări (activări) repetate ale unui subset fixat de task-uri ale programului.

Operarea unui program Giotto se găseşte într-un anumit *mod* la un moment dat. În cadrul unui *mod*, fiecare task este invocat cu o anumită frecvenţă (de unde rezultă perioada fiecărui task).

Durata de execuţie a unui anumit *mod*, denumită *rundă*, rezultă ca cel mai mic multiplu comun al perioadelor task-urilor componente. Operarea unui program Giotto este văzută ca o succesiune de *runde* ale unuia sau mai multor *moduri*. Tranziţia de la un *mod* la altul este efectuată cu ajutorul entităţii *comutator de moduri* (*mode switch*).

(c) *Porturile*: sunt echivalentele logice ale unor locaţii de memorie, destinate comunicaţiei de date şi caracterizate de tipuri bine definite.

Informaţia stocată în fiecare *port* este "persistentă", în sensul că îşi păstrează valoarea până la o nouă actualizare.

Există mai multe categorii disjuncte de porturi:

- *sensor ports*: intrări, actualizate de către mediul înconjurător (senzori);
- *actuator ports*: ieşiri, actualizate de programul Giotto;
- *task ports*: entităţi echivalente cu parametrii de intrare/ieşire ai task-urilor;
- *mode ports*: entităţi echivalente cu parametrii de comunicaţie dintre *modurile* programului Giotto.

Un task poate să-şi menţină o anumită stare de la o execuţie la alta, prin intermediul unui set de *porturi private*, inaccesibile în exteriorul task-ului.

(d) *Driverele*: reprezintă funcţii de conversie a valorilor unor porturi pentru alte porturi. Cu alte cuvinte, *driverele* Giotto sunt secvenţe de cod sistem pentru preprocesarea şi transmiterea datelor între porturile locale ale driverelor.

Driverele pot fi condiţionate cu ajutorul a câte unui predicat care se evaluează pentru un anume *port* de intrare al *driverului*. De exemplu, un task oarecare, invocat de program la un moment dat, nu poate fi executat decât dacă evaluarea condiţiei aplicate *portului de senzor* al task-ului dă rezultate pozitive.

În plan logic, formal, execuţia driverelor se efectuează *instantaneu*, spre deosebire de cea a task-urilor, care are loc pe durata perioadei corespunzătoare. Prin urmare, *driverele* implementează *operaţii atomice*, executate în regim non-preemptiv (cu sistemul de întreruperi dezactivat).

Compilatorul Giotto este o componentă esenţială a metodologiei, având sarcina principală de a genera cod executabil pentru o platformă dată, dintr-un program sursă Giotto care reprezintă o specificare funcţională şi temporală a aplicaţiei. Codul executabil generat conţine trei elemente principale: (i) *codul de funcţionalitate* a task-urilor aplicaţiei ce pot fi scrise de exemplu în C; (ii) *E-codul*, adică instrucţiunile ce definesc comportarea reactivă a task-urilor aplicaţiei, de exemplu timpii de activare şi termenele limită ale task-urilor ca reacţie la evenimentele externe ale sistemului; (iii) *S-codul*, ce conţine o planificare fezabilă a aplicaţiei pe arhitectura ţintă dată.

Pentru a reuşi etapa de compilare a unei aplicaţii Giotto (cu generarea unei planificări fezabile), este necesară calcularea apriori compilării a timpilor maximi de execuţie (WCET) pentru secvenţele de cod funcţional ale task-urilor. Metodologia Giotto nu furnizează în acest sens nici un mecanism care să ajute la obţinerea

timpilor WCET. Pentru uşurarea fazei de compilare a aplicaţiilor Giotto, se pot utiliza biblioteci de tip "run-time" ce furnizează de exemplu nivelul de funcţii şi primitive de planificare şi comunicaţie inter-task pentru o platformă dată. De asemenea, se pot introduce în program adnotări speciale: "Giotto-H" pentru specificarea unor caracteristici şi performanţe hardware ale platformei ţintă, "Giotto-M" pentru specificarea mapării task-urilor pe unităţile de procesare şi a elementelor de comunicaţie pe arhitectura reţelei în cazul sistemelor distribuite, şi "Giotto-S" pentru planificarea task-urilor şi a elementelor de comunicaţie.

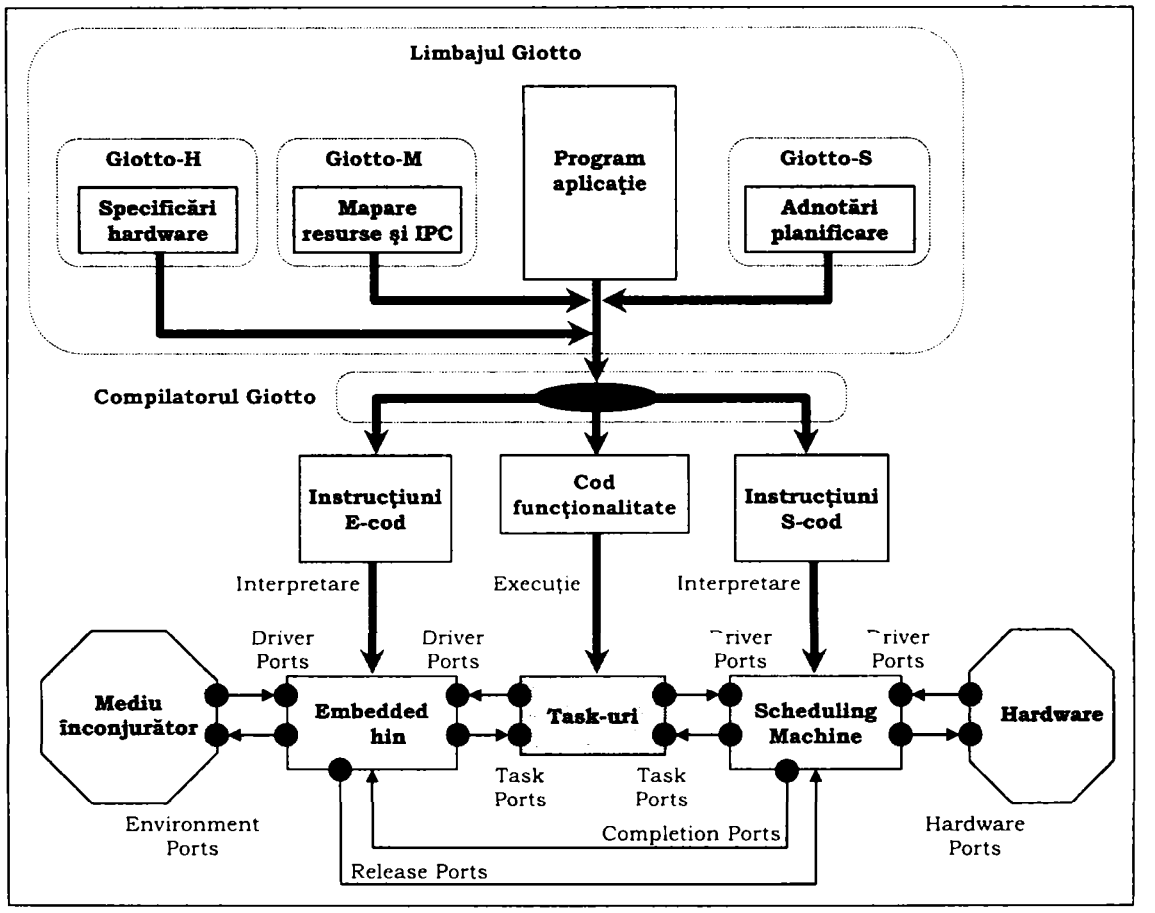

Figura 2-8. Principalele elemente ale metodologiei Giotto

(2) *Mediul (micro-nucleul) de operare*

Metodologia Giotto specifică [Henzinger 02, Kirsch 03, Henzinger 03c] o arhitectură logică particulară pentru execuţia aplicaţiilor timp-real pe diverse platforme ţintă. Mediul de operare este partajat în două maşini virtuale: (a) "E-machine" (Embedded machine), care interpretează şi execută "E-codul", supervizând execuţia "logică" a task-urilor aplicaţiei relativ la evenimentele externe ale sistemului; (b) "S-machine" (Scheduling machine), care interpretează "S-codul" produs de compilator, supervizând execuţia "fizică" a task-urilor aplicaţiei pe baza resurselor sistemului.

Maşina "E" reprezintă componenta reactivă a sistemului de control digital încorporat, independent de resursele hardware sau de constrângerile impuse de planificare. Pe de altă parte, maşina "S" reprezintă componenta "pro-activă" a

sistemului, gestionând execuţia pe procesoarele disponibile a task-urilor active la un moment dat în sistem. Maşina "S" este corespondenta planificatoarelor din sistemele timp-real. Cele două maşini virtuale compun micro-nucleul programabil propus de metodologia Giotto pentru execuţia aplicaţiilor timp-real. Figura 2-8 prezintă principalele elemente ale metodologiei Giotto şi relaţia dintre ele.

2.5.2 Sistemul Spring

Sistemul "Spring" [Stankovic 91a, Stankovic 91b, Stankovic 91c, Stankovic 99] este dezvoltat la Universitatea Massachusetts, Amherst, Statele Unite, pentru a servi ca suport pentru aplicaţii timp real în medii de calcul distribuite.

Arhitectura "SpringNet" se bazează pe o reţea de noduri multiprocesor (vezi Figura 2-9), fiecare nod operând sub nucleul "Spring Kernel" [Stankovic 90]. Nodurile sistemului conţin unul (sau mai multe) procesoare de aplicaţii, unul (sau mai multe) procesoare sistem, un subsistem I/O şi unul pentru comunicaţii. Fiecare procesor dintr-un nod dispune de o memorie locală, ce poate fi accesată şi de către procesoarele sistem.

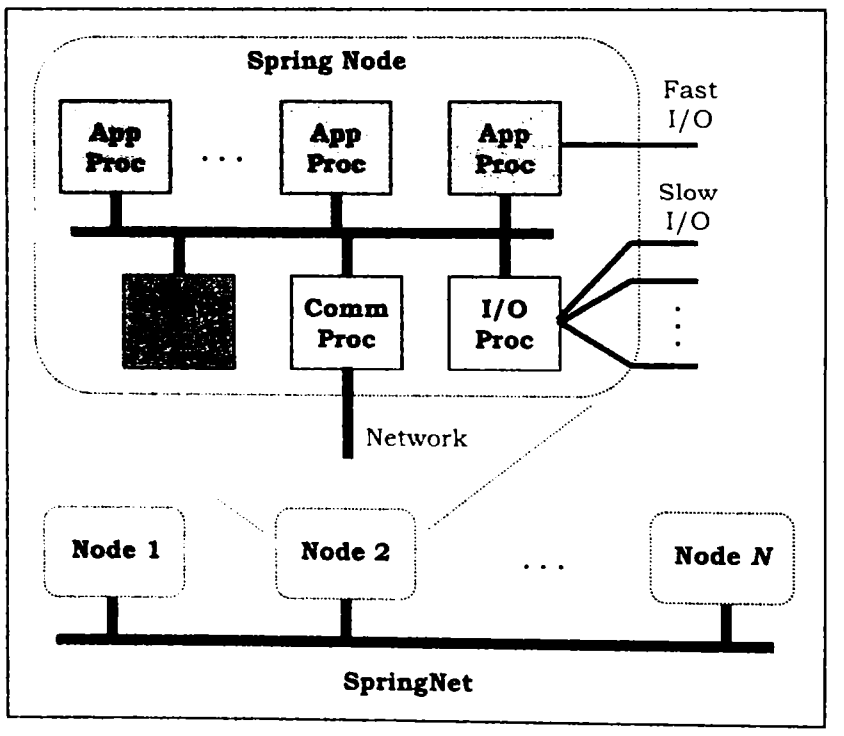

Figura 2-9. Sistemul "SpringNet"

Arhitectura sistemului "Spring" se bazează pe implementarea unui set de idei ce vizează operarea sistemelor timp-real distribuite în condiţii cât mai bune de predictibilitate, flexibilitate şi fiabilitate:

- partiţionare funcţională,
- segmentare/partiţionare a resurselor,
- prealocare selectivă a resurselor,
- garantarea apriori pentru task-urile critice şi garantarea dinamică (on-line) pentru task-urile esenţiale,

- integrarea planificării task-urilor pe procesoare împreună cu alocarea resurselor sistem,
- diferenţierea între constrângerile de timp şi cele de importanţă pentru task-uri,
- utilizarea de informaţii suplimentare în timpul operării sistemului (de exemplu, termene, importanţa task-urilor, cerinţe de fiabilitate, etc.) şi capacitatea de a modifica dinamic aceste informaţii. Aceste proprietăţi conferă *reflectivitate* sistemului.

Partajarea funcţională semnifică faptul că fiecare nod al lui "SpringNet" este un sistem multiprocesor, compus din procesoare aplicaţie care execută task-uri aplicaţie garantate şi planificate apriori, un procesor (sau mai multe procesoare) sistem, care execută algoritmii de planificare şi primitivele sistem, un procesor pentru comunicaţii şi unul sau mai multe procesoare I/O. Procesorul sistem prelevează procesoarele aplicaţie de încărcarea suplimentară impusă de planificare şi de rutinele de operare ale nucleului "Spring", având ca rezultat creşterea semnificativă a vitezei sistemului şi evitarea problemelor de predictibilitate ce pot apărea în execuţia task-urilor garantate, din cauza întreruperilor externe sau a operării sistemului. În cazul defectării procesorului sistem, funcţiile acestuia vor putea fi preluate de unul din procesoarele aplicaţie.

Segmentarea/partajarea resurselor sistemului este realizată în entităţi bine definite ce includ: task-uri şi grupuri de task-uri, segmente de memorie pentru cod, stivă, variabile locale, porturi, discuri virtuale, etc. Atât task-urile şi grupurile de task-uri, cât şi primitivele nucleu sunt limitate ca timp şi ca resurse. Cu alte cuvinte, pentru fiecare task se cunosc: timpul de execuţie (WCET – Worst Case Execution Time) şi cerinţele de resurse (WCRR – Worst Case Resource Requirements) pentru cazurile cele mai defavorabile.

Sistemul reţine şi utilizează o cantitate relativ mare de informaţie cu privire la caracteristicile şi parametrii task-urilor şi ai grupurilor de task-uri. Astfel, pe lângă WCET şi WCRR, se mai pot utiliza parametri ca: nivelul de importanţă, relaţii de precedenţă, cerinţe de fiabilitate, etc. Nucleul utilizează în mod dinamic aceste informaţii pentru a planifica task-urile pentru execuţie şi pentru a rezerva resursele sistem necesare.

Algoritmul de planificare ce stă la baza operării nucleului "Spring" împarte setul de task-uri timp-real sistem şi ale aplicaţiei în task-uri critice (nerespectarea condiţiilor de timp impuse produce consecinţe majore) şi task-uri esenţiale (operarea sistemului şi a aplicaţiei nu este afectată în mod critic în cazul depăşirii termenelor specificate). În cazul task-urilor critice, execuţia acestora este planificată pe procesoarele sistemului iar resursele sunt rezervate apriori, pe toată durata operării. În cazul task-urilor esenţiale, planificarea se face dinamic. Algoritmul încearcă planificarea şi garantarea task-urilor esenţiale adăugând pe rând câte un astfel de task la setul celor garantate până în momentul curent şi verificând dacă noul set de task-uri este fezabil. Principalii parametri consideraţi la planificarea fiecărui task sunt: momentul de apelare, termenul limită sau perioada, WCET, setul task-urilor predecesoare şi necesarul de resurse sistem (utilizate fie în mod exclusiv, fie în mod partajat). Fiind conceput pentru sisteme timp-real distribuite, algoritmul de planificare aplică o euristică ce constă din căutarea iterativă într-un arbore cu planificări posibile ale unui nou task ce se adaugă setului de task-uri garantate.

Această tehnică încearcă practic, la fiecare pas, extinderea planificării curente. Dacă o anumită ramură a arborelui rezultă într-o planificare nefezabilă, se reia căutarea prin tehnici de "backtracking" pe alte ramuri posibile.

În [Niehaus 93, Burleson 93] este propus conceptul interesant de "co-procesor de planificare" (SSCoP – "Spring Scheduling Co-Processor"), care practic implementează hardware algoritmul de planificare al sistemului Spring. Generarea timpilor de planificare pentru task-urile analizate, calcularea funcției de selectare euristică a task-ului următor, revenirea şi controlul tehnicii de "backtracking", etc., sunt operaţii efectuate de blocuri funcționale ale SSCoP, toate acestea fiind controlate de o maşină cu stare finită. Pentru sistemul Spring, SSCoP este în multe privințe similar cu o memorie inteligentă, rapidă şi de dimensiuni mici. Toți regiştrii SSCoP sunt mapați în spațiul de adrese al sistemului, co-procesorul fiind încărcat cu parametrii aferenți task-urilor, iar la terminarea unui ciclu de execuție, se poate citi planificarea obținută (în cazul în care există o planificare fezabilă), constând din lista ordonată a task-urilor împreună cu timpii de execuție asociați.

Rezultatele simulărilor SSCoP demonstrează eficiența acestei soluții [Niehaus 93]: o creştere de peste 30 de ori a vitezei în comparație cu planificatorul software, iar viteza de execuție a secvenței principale din algoritmul de planificare este îmbunătățită cu trei ordine de mărime.

3 Problemele dezvoltării STR stricte pentru aplicaţii critice de achiziţie şi prelucrare numerică de semnal

3.1 Dezvoltarea STR – concepte şi arhitecturi

Marea majoritate a lucrărilor din domeniul sistemelor TR evidenţiază faptul că dezvoltarea sistemelor şi aplicaţiilor TR trebuie să parcurgă o serie de etape (obligatorii) [Mok 83, Stankovic 88, Stankovic 92a, Stankovic 92b, Ostroff 92, Panzieri 93b, Parashkevov 94, Ghosh 94, Ramamritham 94, Chapman 94, Bellini 00, Letia 00]:

1) Concepţie şi proiectare
2) Specificarea funcţională (programare) şi de comportare în timp
3) Verificare/validare formală
4) Analiza temporală de program şi de fezabilitate a execuţiei
5) Generarea codului, a planificării şi implementarea pe o arhitectură ţintă
6) Testarea implementării

Toate etapele dezvoltării STR trebuie să aibă în vedere introducerea timpului ca şi coordonată esenţială a sistemului, şi nu doar ca o caracteristică în funcţie de care să se adapteze sistemul gata dezvoltat [Stankovic 92b]. Tot în această lucrare este evidenţiată dificultatea cerinţelor unei proiectări corespunzătoare a STR, ce derivă din următorul paradox: majoritatea tehnicilor de specificare, proiectare şi verificare se bazează pe abstractizări (care ignoră detaliile de implementare), pe când, pe de altă parte, constrângerile de timp derivă din detalii concrete de implementare şi ale mediului.

În prezent, un număr încă foarte mare de sisteme TR (în special cele stricte) reprezintă practic implementări ad-hoc, puternic orientate pe aplicaţiile vizate şi particularizate pentru arhitecturile implementării. Multe sisteme TR se bazează pe adaptarea şi optimizarea (în sensul creşterii vitezei de reacţie) a unor versiuni de sisteme de operare time-sharing tradiţionale. Un exemplu în acest sens îl reprezintă sistemul *Real-Time Linux*, care derivă din varianta de *UNIX* pentru calculatoare personale – *Linux*, prin instalarea unor pachete soft (*patches*) care modifică unele structuri de date şi rutine ale nucleului, pentru a permite execuţia proceselor ţinând cont de unele specificaţii de timp.

Sistemele de operare tradiţionale se bazează pe noţiunea că task-urile aplicaţiei emit cereri pentru resurse sistem, iar sistemul de operare este proiectat pentru a deservi aceste cereri (în manieră asincronă). Rezultă astfel o comportare bună în situaţiile comune de operare ("average case behavior").

Transformarea unui sistem de operare time-sharing clasic pentru a servi ca platformă pentru aplicaţii TR, se face de regulă adaptând nucleul în scopul creşterii vitezei de reacţie a sistemului [Stankovic 92b]:

- sunt vizate schimbările mai rapide de context;
- nucleul va avea dimensiuni mai mici (cu funcţionalitatea minimală asociată, ca rezultat);
- sistemul va răspunde mai rapid la întreruperi externe;
- se minimizează intervalele în care întreruperile sunt dezactivate;
- sunt introduse şi gestionate fişiere secvenţiale speciale, capabile a acumula date într-un ritm rapid.

Pentru a face faţă cerinţelor temporale impuse de operarea TR, nucleul unui sistem tradiţional va fi adaptat astfel:

- gestionează un ceas de timp-real (*RTC – Real-Time Clock*);
- se oferă un mecanism de planificare bazat pe priorităţi;
- sunt puse la dispoziţia task-urilor aplicaţiilor mecanisme speciale pentru alarme şi contoare de timp cu expirare ("timeouts");
- task-urile pot invoca primitive sistem pentru a stabili întârzieri şi pentru a-şi întrerupe/relua execuţia.

Din punct de vedere al arhitecturilor sistem, în prezent există o multitudine de mecanisme şi concepte special prevăzute pentru a creşte eficienţa şi viteza de procesare a sistemelor. Acestea prezintă însă probleme în ceea ce priveşte predictibilitatea STR, de exemplu îngreunând sau chiar împiedicând calculul timpilor maximi de execuţie (*WCET – Worst Case Execution Time*) ai task-urilor programului [Stankovic 92a, Stankovic 92b, Colnaric 93, Chapman 94]:

- paralelismul operării componentelor interne ale procesoarelor moderne;
- mecanismele de pipeline din procesoare;
- memoriile cache;
- mecanismele de întrerupere;
- utilizarea resurselor partajate;
- memoria virtuală;
- accesarea directă a memoriei (DMA), prin mecanisme tip "cycle stealing";
- magistralele sistem ce permit transferuri asincrone de date (în special pentru sisteme cu resurse partajate).

Tehnicile şi conceptele de tipul celor menţionate mai sus induc o comportare puternic asincronă şi impredictibilă din punct de vedere al timpilor de execuţie pentru sistemele TR, şi trebuie în consecinţă, evitate în faza de proiectare a arhitecturii hardware a STR.

3.2 Predictibilitatea execuţiei task-urilor TR cu termene stricte

Predictibilitatea execuţiei task-urilor în sistemele TR stricte reprezintă o caracteristică şi în acelaşi timp o problemă extrem de importantă. Una dintre consecinţele cele mai importante ale asigurării predictibilităţii unui task oarecare,

reprezintă posibilitatea garantării respectării termenelor şi a comportamentului în timp specificate pentru task-ul respectiv, pe tot parcursul operării sistemului, incluzând situaţiile cele mai defavorabile (de exemplu, în cazuri de încărcare maximă sau de supra-solicitare a resurselor sistemului).

Predictibilitatea are implicaţii directe în conceperea mecanismelor de alocare a resurselor sistem (procesor, memorie, accese I/O, comunicaţii, etc.), impunând necesitatea integrării restricţiilor de timp [Stankovic 92b]. În particular, proiectarea algoritmilor de planificare pentru sistemele TR, trebuie să aibă în vedere asigurarea predictibilităţii de operare, cel puţin pentru task-urile TR cu specificaţii stricte de comportare temporală.

În [Ramamritham 94] se remarcă faptul că pentru a verifica predictibilitatea sistemelor TR (adică, pentru a prezice dacă task-urile îşi vor respecta constrângerile temporale impuse), este necesară efectuarea unei analize a planificabilităţii (sau, o verificare a fezabilităţii) sistemului. Pentru cazul sistemelor TR stricte, unde respectarea specificaţiilor temporale este o cerinţă esenţială, ce trebuie garantată în orice condiţii de operare, faza de analiză a planificabilităţii trebuie efectuată înaintea lansării în execuţie a sistemului (aplicaţiei). Prin urmare, o etapă esenţială în dezvoltarea aplicaţiilor TR critice este *analiza offline* a fezabilităţii acestora.

O problemă majoră din punctul de vedere al predictibilităţii este utilizarea neîngrădită a întreruperilor [Stewart 01]:

- Întreruperile sunt cea mai frecventă cauză a fenomenului de inversare a priorităţilor (*priority inversion*) dintr-un sistem cu planificare pe bază de priorităţi: execuţia unui task cu prioritate mai mare ce va accesa o anumită resursă partajată, este blocată de un task cu prioritate mai mică ce a fost întrerupt de către primul, în timpul accesării aceleiaşi resurse.
- Rutinele de tratare a întreruperilor (*interrupt handlers*) pot întrerupe orice task din sistem, indiferent de prioritatea acestuia. Pe de altă parte, aceste rutine nu sunt (nu pot fi) planificate, corespunzând de regulă apariţiei unor evenimente asincrone în sistem.
- Rutinele de tratare a întreruperilor sunt executate în contexte diferite de cele ale task-urilor aplicaţiei, forţând astfel utilizarea variabilelor globale pentru a putea schimba informaţie cu restul aplicaţiei.
- Din punct de vedere al testării aplicaţiilor, întreruperile sunt dificil de simulat şi de testat, pe de o parte din cauză că uneltele soft de testare permit foarte rar setarea de puncte de control a programului în interiorul handlerelor, iar pe de altă parte, din cauza asincronismului apariţiei întreruperilor.

Se impune în consecinţă, mai ales în cazul sistemelor TR stricte, minimizarea pe cât posibil a utilizării întreruperilor. In extremis, [Stewart 01] subliniază faptul că un sistem TR ideal nu prezintă nici o întrerupere.

Capitolul 5 al lucrării conţine o discuţie şi mai detaliată a avantajelor şi dezavantajelor celor două abordări – preemptivă şi non-preemptivă. Tot aici va fi prezentată o clasă de aplicaţii TR stricte, ce conţin task-uri cu execuţie fixă în cadrul perioadei şi a căror planificare nu poate fi rezolvată cu algoritmi preemptivi.

3.3 Specificarea, programarea şi analiza temporală a aplicaţiilor TR critice

În prezent, există o varietate mare de metode de specificare şi verificare formală a sistemelor şi aplicaţiilor TR, din care însă, nu a reuşit nici una să fie acceptată în întregime de mediul academic şi industrie. Standardizarea tehnicilor şi modelelor utilizate în dezvoltarea sistemelor TR reprezintă una din problemele importante din domeniu.

Pe de altă parte, majoritatea metodelor de specificare formală utilizate pentru sistemele TR derivă din abordări clasice, care nu pun un accent deosebit pe coordonata timp [Chapman 94, Parashkevov 94]: Time Petri Nets, Timed Petri Nets (bazate pe reţelele Petri), Timed CSP, CSP+T, CCSR, Timed CCS (bazate pe algebre de proces), etc. De aici rezultă o serie de inconveniente:

- Multe metode se bazează pe conceptul de timp global (şi de ceasuri sincronizate global, în cazul STR distribuite). În practică, această premisă este foarte dificil, dacă nu imposibil, de implementat.
- Fiind derivate din tehnici ne-temporale, metodele de specificare formală au posibilităţi de expresie limitate de predecesorii lor.
- Există un decalaj important între rezultatele cercetărilor din domeniul metodelor formale de timp-real şi implementările practice din industrie.

Din cele de mai sus se desprinde necesitatea conceperii unor unelte puternice şi intuitive, destinate specificării incrementale şi verificării automatizate a proiectelor sistemelor TR.

Faza ce urmează specificării şi verificării formale a STR – programarea aplicaţiilor – necesită, la rândul ei, dezvoltarea unor limbaje specializate, cu următoarele caracteristici principale [Parashkevov 94]:

- dependenţa cât mai redusă faţă de hardware şi sistemul de operare a platformelor ţintă;
- trebuie să posede facilităţile limbajelor moderne de programare (set puternic de tipuri de date, programare structurată, modularitate, interfeţe bine definite, posibilitatea compilării separate a modulelor, etc.);
- trebuie să ofere primitivele necesare accesării şi controlului elementelor hardware ca registri, adrese I/O, întreruperi, etc.;
- furnizarea de suport natural pentru exprimarea explicită a concurenţei, a comunicării şi sincronizării proceselor aplicaţiei;
- trebuie să ofere programatorului mecanismele necesare descrierii explicite a proprietăţilor şi constrângerilor temporale ale codului programelor: execuţie periodică, contoare de timp, termene pentru task-uri şi grupuri de task-uri, întârzieri, etc.;
- restricţionarea sau chiar interzicerea construcţiilor de program şi a structurilor de date ce nu sunt limitate în timp şi spaţiu: recursivitatea, structuri de memorie dinamică, bucle cu un număr nelimitat sau necunoscut de iteraţii la momentul compilării, etc.;
- trebuie să ofere informaţii suplimentare utilizate în analiza planificabilităţii codului compilat.

3.4 Modelul unui STR strict pentru aplicaţii critice

Sintetizând cerinţele şi problemele enumerate în paragrafele anterioare, se poate defini un model generic, ce poate fi urmărit în conceperea tehnicilor şi metodelor de dezvoltare a sistemelor TR stricte, orientate pe aplicaţii critice de achiziţie şi prelucrare numerică a semnalelor (DSP-based systems) şi de control digital încorporat (embedded systems). Modelul STR strict prezintă următoarele caracteristici de bază:

- Definirea clară a mai multor faze necesare în dezvoltarea STR: specificarea, proiectarea, verificarea/validarea, analiza de program şi analiza fezabilităţii.
- Timpul, ca şi coordonată esenţială a operării sistemului, va trebui introdus ca parametru cheie în toate fazele dezvoltării STR.
- Metodele cu care se abordează fazele dezvoltării STR trebuie să fie unitare şi omogene.
- Asigurarea predictibilităţii execuţiei task-urilor TR stricte ale sistemului reprezintă o cerinţă esenţială, ce afectează toate etapele dezvoltării şi implementării.
- Algoritmii de planificare trebuie atent concepuţi astfel încât să asigure execuţia predictibilă, cu respectarea termenelor impuse task-urilor TR stricte şi conferind sistemului, pe de altă parte, eficienţă şi flexibilitate cât mai mari.
- Programarea task-urilor aplicaţiilor TR stricte trebuie realizată cu ajutorul unor limbaje care să permită exprimarea de constrângeri temporale şi totodată să permită analiza temporală a programelor (determinarea timpilor WCET, BCET, etc.).
- Pentru asigurarea predictibilităţii, următoarele mecanisme şi concepte sistem trebuie evitate:
  - memorii cache
  - memorii virtuale
  - transferurile DMA (mai ales cele de tipul "cycle stealing")
  - magistralele sistem ce permit transferuri asincrone de date
- O atenţie specială trebuie acordată utilizării (limitate) a următoarelor mecanisme:
  - întreruperile (conferă sistemului un comportament puternic asincron şi, deci, impredictibil)
  - pipelining
- Sistemul gestionează un ceas de timp-real propriu, care defineşte domeniul temporal sistem.
- Comunicarea inter-task-uri trebuie să utilizeze mecanisme speciale, predictibile, care să evite operarea asincronă.
- Accesul la resurse partajate va trebui controlat şi rezolvat prin mecanisme de sincronizare predictibile, sau, dacă se poate, chiar evitat.

SECŢIUNEA II. CONTRIBUŢII LA PROIECTAREA ŞI DEZVOLTAREA *STR* STRICTE PENTRU APLICAŢII CRITICE

Capitolele secţiunii de faţă conţin contribuţiile aduse în domeniul proiectării şi dezvoltării sistemelor timp-real stricte. Secţiunea reprezintă astfel fundamentul teoretic al tezei, ce se bazează pe ideea introducerii mecanismelor non-preemptive în proiectarea, analiza şi dezvoltarea STR stricte pentru aplicaţii critice de APNS şi control digital încorporat.

Capitolul 4 introduce şi studiază setul omogen de modele pentru timpul real, semnale şi task-uri, ce constituie baza proiectării şi dezvoltării STR stricte pentru aplicaţii critice. Modelul task-urilor TR stricte, ModX-ul, este prezentat şi analizat în detaliu.

Planificarea aplicaţiilor TR stricte este abordată în Capitolul 5, accentul fiind pus pe tehnicile de planificare non-preemptivă, capabile să garanteze operarea cu maximă predictibilitate a STR stricte, în orice condiţii. Sunt studiaţi doi dintre algoritmii cei mai utilizaţi în prezent pentru planificarea non-preemptivă şi sunt introduse o serie de adaptări şi de noi algoritmi, în funcţie de diversele cerinţe impuse de tipul aplicaţiilor timp-real ce se doresc implementate.

Capitolul 6 prezintă analiza performanţelor algoritmilor de planificare discutaţi în această secţiune. Evaluarea performanţelor a fost realizată pe baza unor programe de test special dezvoltate şi executate pe cele 12 staţii de calcul ale laboratorului DSPLabs, rezultând un total de peste 150000 de teste.

Modelul unui sistem complet (al unei metodologii), OPEN-HARTS, care unifică toate conceptele introduse şi discutate în această secţiune, este prezentat în Capitolul 7. Sistemul OPEN-HARTS este conceput ca un set de soluţii pentru problemele ridicate de proiectarea şi dezvoltarea unitară a sistemelor şi aplicaţiilor TR stricte.

4 Informaţia de timp, evenimente şi acţiuni. Modelul task-ului TR strict

Timpul, ca şi coordonată esenţială a STR stricte, trebuie să fie implicat în toate fazele de dezvoltare: specificare şi verificare formală, programare, analiză, planificare şi execuţie.

Capitolul de faţă analizează informaţia necesară pentru a descrie comportarea în domeniul timp a elementelor unui STR strict şi utilizarea acesteia în toate fazele dezvoltării sistemului. Analiza are ca rezultat extragerea unui set minim necesar de parametri temporali (predicate), suficient de intuitiv pentru a fi utilizat cu uşurinţă în specificarea şi programarea sistemului şi cu ajutorul cărora se construieşte în final un model al task-ului TR strict.

4.1 Timpul sistem şi un model temporal pentru dezvoltarea STR stricte

Funcţionarea sistemelor de calcul în general, şi a STR în particular, se bazează pe o frecvenţă de tact generată de un oscilator intern sau extern. Frecvenţa de tact defineşte aşa-numitul "ciclu procesor" sau "ciclu instrucţiune" al unităţii de procesare a sistemului, reprezentând durata execuţiei unei instrucţiuni procesor (exprimată în secunde, Hz sau perioade de tact). În acelaşi timp, frecvenţa de tact alimentează baza de timp a STR, anume ceasul de timp-real (*RTC, real-time clock*). Prin urmare, din punctul de vedere al STR, granularitatea de timp minimă cu care sistemul poate lucra este egală cu durata unui ciclu procesor.

Exemplul 4-1.

Pentru exemplificare, să considerăm cazul procesorului numeric de semnale Motorola DSP56307 [Motorola 98, Motorola 99, Motorola 00], care, pentru o aplicaţie oarecare, operează la o frecvenţă de tact,

$$CoreCLK = 32 \text{ MHz}.$$

Ciclul procesor pentru DSP56307 este egal cu o perioadă de tact, rezultând astfel granularitatea de timp minimă ca fiind egală cu durata ciclului procesor, adică

$$\Delta t_{CLK} = 31.25 \text{ ns}.$$

Mai departe, în cazul în care se utilizează un RTC cu un factor de divizare (prescalare) de 4, unitatea de timp măsurată de către RTC este:

$$\Delta t_{RTC} = 4 \cdot \Delta t_{CLK} = 125 \text{ ns}.$$

❑

Concluzia celor de mai sus este că sistemele digitale de calcul, incluzând şi STR, operează cu valori discrete de timp. STR utilizează ca bază de timp pentru tratarea evenimentelor şi a task-urilor, *ceasul de timp-real*, *RTC*.

Utilizarea RTC a cărui arhitectură uzuală include un contor de timp, defineşte domeniul temporal de operare al sistemelor TR. Astfel,

$$t_0 = 0$$

poate fi identificat cu momentul startării sistemului (cuplarea la tensiune şi iniţializarea contorului RTC), fiind denumit "momentul iniţial".

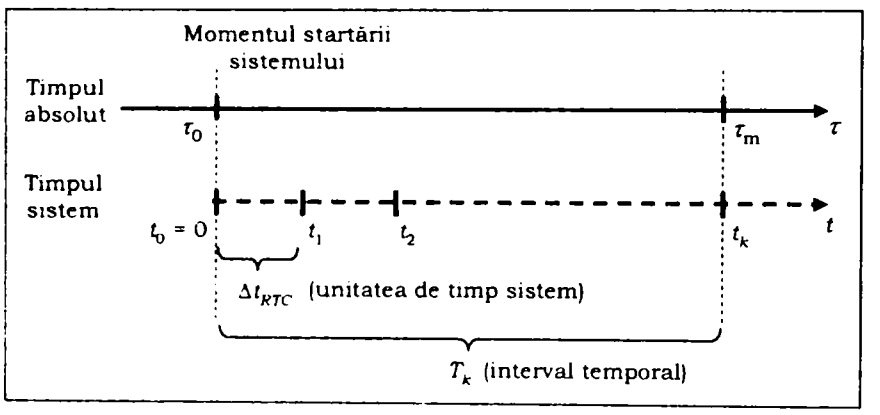

Figura 4-1. Timpul sistem şi timpul absolut pentru STR

În Figura 4-1 sunt reprezentate cele două domenii temporale: timpul sistem şi cel absolut, la care se raportează. Variabilele de timp au fost notate distinct, τ pentru timpul absolut şi t pentru timpul sistem. La momentul τ_0 are loc startarea STR (deci, $t_0 = 0$).

Oricare două instanţe succesive ale timpului sistem, t_i şi t_{i+1} sunt distanţate cu Δt_{RTC} în domeniul temporal absolut, şi cu 1 în domeniul temporal al sistemului. Două evenimente care apar în intervalul Δt_{RTC} sunt văzute ca fiind simultane de către sistem. De aici rezultă valoarea unităţii temporale sistem şi faptul că aceasta permite existenţa metricilor temporale.

Lungimea unui interval temporal oarecare T_k poate fi determinată astfel:

$$\begin{cases} T_k = \tau_m - \tau_0 = k \cdot \Delta t_{RTC} & \text{pentru timpul absolut} \\ T_k = t_k - t_0 = k & \text{pentru timpul sistem} \end{cases} \quad (4\text{-}1)$$

Legătura dintre cele două domenii temporale este sintetizată de relaţia de corespondenţă a unei instanţe de timp (un punct din cele două domenii):

$$\tau_i = \tau_0 + (t_i - t_0) \cdot \Delta t_{RTC} \quad (4\text{-}2)$$

Importanţa relaţiei (4-2) şi a utilizării ambelor modele temporale derivă din faptul că în faza de specificare a comportamentului temporal al STR, utilizatorul/programatorul de aplicaţii judecă evenimentele din perspectiva timpului absolut, pe când sistemul TR operează în domeniul temporal discret, propriu. În faza imediat următoare din cadrul procesului de dezvoltare a sistemului/aplicaţiei TR, anume în faza de verificare şi validare, este necesară convertirea în mod corespunzător a specificaţiilor, din modelul temporal absolut în cel sistem.

În concluzie, modelul temporal propus pentru dezvoltarea STR stricte are următoarele caracteristici principale (vezi Capitolul 2):

- Domeniul temporal are o structură liniară, discretă;
- Existența momentului inițial: $t_0 = 0$ (adică domeniul temporal este limitat la stânga);
- Domeniul temporal sistem poate fi echivalat cu mulțimea numerelor naturale:

$$t \in \mathrm{N} \tag{4-3}$$

- Momentul inițial corespunde startării STR, adică momentului τ_0 din domeniul temporal absolut;
- Unitatea de timp corespunde unui interval de lungime Δt_{RTC} din domeniul temporal absolut;
- Timpul sistem permite exprimarea de relații cantitative între instanțe individuale (puncte) sau între intervale temporale. Astfel, modelul temporal prevede existența unei metrici pentru timp;
- Cu ajutorul metricii pentru timp se poate determina lungimea unui interval temporal, cu (4-1);
- Corespondența unei instanțe de timp din domeniul temporal sistem în cel absolut se poate determina cu (4-2).

Deoarece modelul temporal propus permite operarea atât cu instanțe punctuale de timp cât şi cu intervale temporale, pentru clarificare, introducem următoarele notații care vor fi utilizate în continuare în lucrare:

Tabelul 4-1. Notații ale entităților temporale utilizate în modelul propus

Entitate	Notație	Semnificație
Instanță de timp	t	Variabila de timp sau parametru. Utilizat ca parametru, identifică un anumit moment de timp (punct în domeniul temporal). Distincția dintre domeniul temporal absolut şi domeniul temporal sistem se va face explicit, acolo unde e cazul: • $t \in \mathbf{N}$, pentru modelul temporal sistem, • $t \in \mathbf{R}$, pentru modelul temporal absolut.
Interval temporal	$T = [t_{sT}, t_{eT}]$	Este caracterizat printr-o instanță de început (t_{sT}) şi una sfârşit (t_{eT}), care pot fi precizate (exprimare explicită, absolută) sau nu (exprimare implicită, relativă). Utilizat pentru a identifica un anumit interval temporal cât şi pentru a exprima lungimea (durata) intervalului.

4.2 Evenimente în sisteme TR stricte

Evenimentele sunt definite ca o modalitate prin care diverşi agenţi (în cazul nostru, STR, utilizatorul/programatorul de aplicaţii TR, etc.) clasifică anumite tipare semnificative de schimbare [Allen 94]. Există destul de puţine restricţii în stabilirea semanticii evenimentelor, cu excepţia că acestea trebuie să implice cel puţin un obiect într-un interval de timp sau cel puţin o schimbare de stare. Diferenţa dintre "evenimente" şi "stări" poate fi exprimată intuitiv şi prin faptul că primele descriu schimbări pe când celelalte descriu aspecte care nu se schimbă. Cu alte cuvinte, se poate spune că evenimentele "apar" şi stările "se menţin".

Într-o accepţiune mai restrânsă şi din punctul de vedere al specificării şi operării STR prezentat în lucrarea de faţă, evenimentele sunt descrise prin *semnale* ce determină schimbări de stare ale sistemului.

Semnalele pot fi clasificate după mai multe criterii, trei dintre acestea, care ni se par de interes în problema dezvoltării STR stricte, fiind:

a) După modul (rata) apariţiei semnalelor:
  a.1) semnale periodice
  a.2) semnale sporadice (aperiodice)

b) După numărul de apariţii:
  b.1) semnale cu apariţie singulară
  b.2) semnale cu apariţie temporară
  b.3) semnale cu apariţie permanentă

c) După sursa semnalelor, raportată la STR:
  c.1) semnale de intrare
  c.2) semnale de ieşire
  c.3) semnale interne

În cele ce urmează vom analiza toate aceste tipuri de semnale, extrăgând principalele caracteristici cu privire la comportarea în timp. Rezultatul analizei va sta la baza construirii unui model minimal, intuitiv al semnalelor care să poată fi utilizat în fazele de specificare, verificare/validare şi analiză a sistemelor TR în general, şi a STR stricte în particular.

a.1) *Semnale periodice*

Semnalele periodice sunt caracterizate din punct de vedere al comportamentului în timp prin faptul că au o perioadă cunoscută. Notăm perioada unui semnal oarecare S_i cu $T_{pr}^{S_i}$. Dacă cea de-a k apariţie a lui S_i are loc la momentul t_k, atunci cea de-a $(k+1)$ apariţie va avea loc la momentul

$$t_{k+1} = t_k + T_{pr}^{S_i} \tag{4-4}$$

Exemple de semnale periodice sunt liniile de tact din comunicaţiile sincrone, liniile de date şi control din sistemele de achiziţie a datelor, etc.

a.2) *Semnale aperiodice (sporadice)*

Semnalele aperiodice se caracterizează ca şi comportare în timp prin faptul că intervalele de timp dintre mai multe apariţii consecutive nu sunt egale. Aşadar, comportamentul în timp al semnalelor aperiodice este mult mai puţin restricţionat în comparaţie cu semnalele periodice. În cazul sistemelor şi al mediilor deterministe (cazul considerat în lucrarea de faţă), se poate însă stabili sau determina un interval de timp minim, notat cu $T_{pr}^{S_i}$, care separă oricare două apariţii consecutive ale unui semnal aperiodic S_i. Dacă cea de-a k apariţie a lui S_i are loc la momentul t_k, atunci cea de-a $(k+1)$ apariţie va avea loc la momentul

$$t_{k+1} \geq t_k + T_{pr}^{S_i} \tag{4-5}$$

Se observă că relaţiile (4-4) şi (4-5), ce specifică momentul unei noi apariţii a semnalului periodic, respectiv aperiodic, S_i, faţă de instanţa ultimei apariţii, sunt asemănătoare. Această observaţie permite abordarea (tratarea) celor două tipuri de semnale într-o manieră unitară, aşa cum se va vedea în subcapitolul următor.

Exemple de semnale aperiodice sunt comunicaţiile asincrone, interfeţele cu operatorul, subsistemele de comandă cu bucle de reacţie şi reglare, interfeţe intrare-ieşire ce operează pe bază de întreruperi, etc.

b.1) *Semnale cu apariţie singulară*

Semnalele cu apariţie singulară sunt în general semnale care startează sau opresc diverse procese. În cazul acestora, se pune problema comportării sistemului *până* în momentul apariţiei semnalului şi *după* apariţie. Apariţia în sine semnalului este un eveniment asincron. Tratarea acestui tip de semnale va fi detaliată în subcapitolul următor.

Exemple de semnale cu apariţie singulară sunt comenzile utilizator sau sistem, de tip "start/stop", comenzile de schimbare a modului de operare, etc.

b.2) *Semnale cu apariţie temporară*

Semnalele cu apariţie temporară caracterizează anumite procese ce se desfăşoară pe un interval de timp limitat de-a lungul operării sistemelor şi a interacţiunii acestora cu mediul. Faptul că un anumit semnal S_i are apariţie temporară poate fi caracterizat prin durata intervalului temporal în care au loc evenimentele apariţiei semnalului (durata intervalului în care semnalul este "activ", $T_{act}^{S_i}$). În cazul în care semnalul este şi periodic, durata intervalului de apariţii poate fi echivalată cu numărul de apariţii:

$$N^{S_i} = \frac{T_{act}^{S_i}}{T_{pr}^{S_i}}$$

unde $T_{pr}^{S_i}$ este perioada semnalului S_i.

Se poate observa că semnalele cu apariţie singulară (b.1) pot fi tratate ca un caz particular de semnale cu apariţie temporară, în condiţiile în care numărul de apariţii este egal cu 1.

Exemple de semnale cu apariţie temporară sunt interfeţele cu operatorul care permit comandarea mai multor moduri disjuncte de operare ale sistemului.

b.3) *Semnale cu apariţie permanentă*

Semnalele cu apariţie permanentă caracterizează procesele care se desfăşoară pe întreaga durată a operării sistemelor. Acest tip de semnale pot fi considerate la rândul lor un caz particular de semnale cu apariţie temporară, în condiţiile în care numărul de apariţii şi durata intervalului de apariţii sunt infinite.

Exemple de semnale cu apariţie permanentă se întâlnesc în cazul sistemelor de achiziţie numerică de semnal care operează în mod continuu (de exemplu, sistemele de achiziţionare a parametrilor meteorologici din staţiile meto, termometre digitale, sisteme de control digital al operării automate a uşilor, etc.).

c) *Clasificarea după sursa semnalelor, raportată la STR*

Figura 4-2 exemplifică cele trei tipuri de semnale, clasificate după sursa acestora: semnale de intrare (S_5), de ieşire (S_6) şi semnale interne (S_1, S_2, S_3 şi S_4).

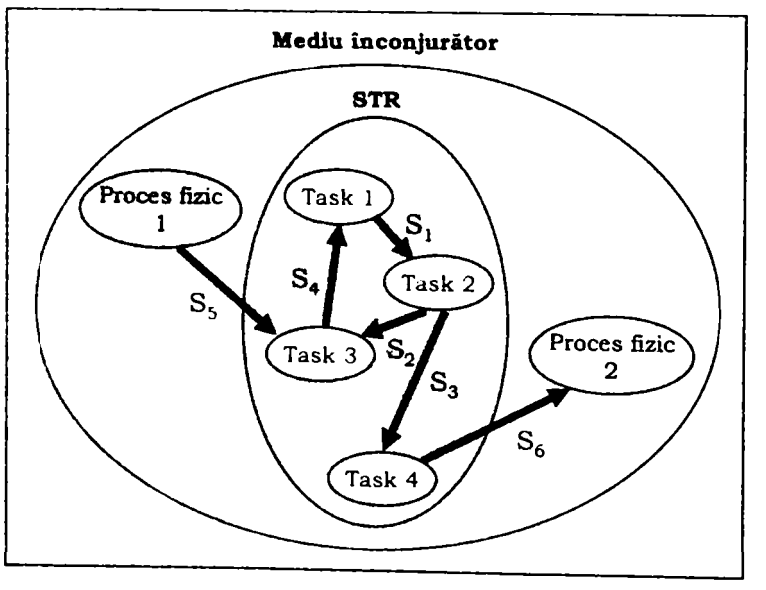

Figura 4-2. Ilustrare a celor trei tipuri de semnale: de intrare, de ieşire şi interne

c.1) *Semnale de intrare*

Semnalele de intrare au ca sursă mediul înconjurător în cadrul căruia operează STR. Faţă de aceste semnale, STR se comportă ca un sistem reactiv: apariţia semnalului de intrare este urmată de tratarea acestuia în cadrul sistemului (cu ajutorul unui task sau al unui set de task-uri specifice). De regulă, rezultatele tratării semnalelor de intrare se concretizează în alte semnale – interne sau de ieşire, reprezentând reacţia STR.

Tratarea semnalelor de intrare este o problemă deosebit de importantă pentru dezvoltarea şi operarea STR stricte, şi modul în care este rezolvată diferenţiază diferitele arhitecturi şi concepte de sisteme TR. Problema va fi abordată în detaliu în subcapitolul următor.

c.2) *Semnale de ieşire*

Semnalele de ieşire sunt semnalele prin intermediul cărora STR acţionează asupra mediului înconjurător. Relativ la aceste semnale, STR se comportă ca un sistem interactiv, de tip "master". Toate caracteristicile semnalelor sunt stabilite de către sistemul care le generează.

c.3) *Semnale interne*

Semnalele interne sunt rezultatul interacţiunii dintre task-urile sistemului şi constau de regulă din: semnale de sincronizare şi de notificare, şi semnale ce conţin informaţie schimbată între task-uri (comunicaţia de date şi parametri, *IPC – Inter-Process Communication*). Caracteristicile temporale ale semnalelor interne sunt dictate de caracteristicile de comportare în timp a task-urilor implicate.

Tabelul 4-2 oferă o reprezentare sintetică a modelului semnalelor prin descrierea principalilor parametri de comportare temporală ai semnalelor, discutaţi până acum. Parametrii din tabel vor fi utilizaţi pentru modelarea semnalelor în faza de specificare a STR, urmând a avea implicaţii în definirea comportării temporale ale task-urilor sistem.

Tabelul 4-2. Modelul semnalelor pentru specificarea STR

Parametru	**Descriere**	**Categorie de semnale**
$T_{pr}^{S_i}$	Perioada semnalului S_i.	Semnale periodice
	Durata minimă de timp ce separă oricare două apariţii consecutive ale semnalului S_i.	Semnale aperiodice (sporadice)
$T_{act}^{S_i}$	Durata intervalului în care au loc apariţii ale semnalului S_i.	Semnale temporare
N^{S_i}	Numărul de apariţii ale semnalului S_i.	
	$N^{S_i} = 1$	Semnale singulare
	$N^{S_i} = n$, finit	Semnale temporare
	$N^{S_i} = \infty$	Semnale permanente

4.3 Tratarea evenimentelor: acţiuni TR stricte

Într-o accepţiune generică, acţiunile sunt definite ca activităţi pe care un sistem (obiect, persoană, etc.) le desfăşoară în anumite condiţii [Allen 94]. Din perspectiva tematicii abordate în lucrarea de faţă, acţiunile sunt văzute în strânsă relaţie de cauzalitate cu evenimentele. Cu alte cuvinte, apariţia unui anume eveniment sau a unei secvenţe bine definite de evenimente, provoacă o acţiune, şi reciproc, o acţiune anume, provoacă apariţia unui eveniment sau a unui set bine precizat de evenimente. Relaţia de cauzalitate eveniment ↔ acţiune este în strânsă concordanţă cu accepţiunea noastră de STR determinist (care operează într-un mediu determinist) şi este de asemenea în legătură directă cu proprietatea de sistem reactiv (sau reflex) – proprietate importantă în cazul STR stricte.

Din perspectiva STR, o acţiune este reprezentată printr-un task sau o secvenţă bine definită de task-uri specifice (subgraf de task-uri). Relaţia de cauzalitate dintre evenimente (semnale) şi acţiuni (task-uri) implică faptul că unele proprietăţi ale semnalelor determină proprietăţi similare pentru task-uri, şi reciproc, mai ales în ceea ce priveşte comportamentul temporal în cazul STR.

Înainte de a aborda problema tratării semnalelor de către task-urile unui STR se impune introducerea unui model temporal preliminar al task-ului TR.

Definiţia 4-1.

Un *task* (notat M_i) reprezintă o secvenţă de instrucţiuni pe care sistemul o execută în scopul desfăşurării unor acţiuni. Execuţia task-urilor în cadrul sistemelor TR se face într-o manieră *planificată*, ţinând cont de coordonata timp şi de caracteristicile temporale ale task-urilor.

❑

Fiecare task al unui STR este *invocat* de către sistem pentru a fi executat, fie ca răspuns al sistemului la un eveniment sau la un set de evenimente, fie pentru a se obţine apariţia unui eveniment sau a unui set de evenimente. Notăm *momentul de invocare* (*invocation time, request time*) al task-ului M_i cu $t_{rq}^{M_i}$.

Task-urile unui STR au, de regulă un număr oarecare (mai mare sau egal cu 1) de execuţii. Notăm cu N^{M_i} numărul total de execuţii ale task-ului M_i ($N^{M_i} \geq 1$). Fiecare ciclu k de execuţie începe din momentul invocării $t_{rq,k}^{M_i}$, şi se termină la momentul invocării pentru ciclul următor, $t_{rq,k+1}^{M_i}$.

Definiţia 4-2.

Definim *durata de execuţie* a task-ului M_i (*execution interval*) pentru ciclul k de execuţie, notată cu $T_{ex,k}^{M_i}$, intervalul de timp (*nenul*) necesar execuţiei lui M_i în cadrul sistemului (intervalul de timp în care procesorul sistemului este alocat execuţiei lui M_i).

❑

Aşa cum rezultă din Capitolele 2 şi 3 ale lucrării, durata de execuţie a unui task variază de la o execuţie la alta. Aşadar, $T_{ex,k}^{M_i}$ nu este constantă, depinzând de o serie de factori, cum ar fi valoarea momentană a informaţiei preluată de la task-urile executate anterior şi de calea pe care o urmează execuţia în cadrul secvenţelor de instrucţiuni ale task-ului (de exemplu, în cazul instrucţiunilor de ramificare de tip "for" sau "case"), etc. În cazul sistemelor TR şi în particular, al STR stricte, faptul că durata de execuţie a task-urilor nu e constantă de la o execuţie la alta constituie o problemă dificilă, în special pentru proiectarea şi implementarea algoritimilor de planificare (vezi Capitolul 3). Vom reveni la acest subiect important în capitolele următoare ale lucrării.

Definiţia 4-3.

Numim *perioadă* a task-ului M_i (*period*), notată cu $T_{pr}^{M_i}$, durata minimă a unui ciclu de execuţie al lui M_i, adică intervalul minim de timp definit de oricare două momente de invocare consecutive ale task-ului M_i:

$$T_{pr}^{M_i} = \min_{1 \le k \le N^{M_i}} \left\{ \left(t_{rq,k+1}^{M_i} - t_{rq,k}^{M_i} \right) \right\} \tag{4-6}$$

unde $k \in \{1, \ldots, N^{M_i}\}$ reprezintă numărul ciclului curent de execuţie.

❑

Se observă că Definiţia 4-3 şi relaţia (4-6) permit atât existenţa task-urilor periodice, cât şi a celor aperiodice (sporadice) ale unui STR. Acestea pot fi definite într-o manieră unitară, asemănătoare cu cea a semnalelor periodice şi aperiodice (relaţiile (4-4), (4-5) şi [Jeffay 91]).

Definiţia 4-4.

Fie $t_{rq,k}^{M_i}$ cel de-al k-lea moment de invocare al task-ului M_i (adică, momentul ce defineşte cel de-al k-lea ciclu de execuţie al lui M_i). M_i se numeşte *task periodic*, de perioadă $T_{pr}^{M_i}$, dacă prezintă o comportare în timp ce respectă următoarele reguli:

i) Cea de-a $(k + 1)$ invocare a lui M_i apare exact la momentul:

$$t_{rq,k+1}^{M_i} = t_{rq,k}^{M_i} + T_{pr}^{M_i} \tag{4-7}$$

ii) Cea de-a k-a execuţie a lui M_i trebuie să înceapă nu mai devreme de $t_{rq,k}^{M_i}$ şi să se termine nu mai târziu de $t_{rq,k}^{M_i} + T_{pr}^{M_i}$.

❑

Reformulând Definiţia 4-4, spunem că task-ul periodic M_i solicită $T_{ex,k}^{M_i}$ unităţi din timpul procesor în fiecare interval $\left[t_{rq,k}^{M_i}, t_{rq,k}^{M_i} + T_{pr}^{M_i} \right]$.

Definiţia 4-5.

Fie $t_{rq,k}^{M_i}$ cel de-al k-lea moment de invocare al task-ului M_i. M_i se numeşte *task aperiodic (sporadic)*, dacă prezintă o comportare în timp ce respectă următoarele reguli:

i) Cea de-a $(k + 1)$ invocare a lui M_i apare la momentul:

$$t_{rq,k+1}^{M_i} \geq t_{rq,k}^{M_i} + T_{pr}^{M_i} \tag{4-8}$$

ii) Cea de-a k-a execuţie a lui M_i trebuie să înceapă nu mai devreme de $t_{rq,k}^{M_i}$.

❑

Un sistem poate conţine un număr oarecare de task-uri, ce pot fi *independente* sau *dependente* unul faţă de celălalt. Dependenţele dintre task-urile unui STR sunt de două tipuri: (i) *dependenţă de control* (*control dependence*, *control flow*) şi (ii) *dependenţă de date* (*data dependence*, *data flow*), care pot fi grupate formal cu ajutorul aşa-numitei *dependenţe de program* (*program dependence*). Dependenţele de program ale task-urilor impun restricţii (de multe ori, dificil de rezolvat) pentru planificarea în execuţie a acestora. Un task M_j nu poate fi planificat şi executat la un moment oarecare t decât în condiţiile în care sunt îndeplinite *toate* dependenţele sale de program, semnificând de exemplu, ca task-ul M_i să fi obţinut prin execuţie, până la momentul $(t - 1)$, valorile parametrilor săi de ieşire din care o parte sunt transmişi lui M_j ca parametri de intrare.

Definiţia 4-6.

Definim *momentul gata de execuţie* al task-ului M_i (*ready time*) pentru ciclul k de execuţie, notat cu $t_{rd,k}^{M_i}$, instanţa de timp la care task-ul M_i îndeplineşte condiţiile de execuţie şi poate fi planificat pentru execuţie în ciclul k, în cadrul sistemului TR.

❑

Sistemele TR stricte impun termene limită pentru execuţia task-urilor. În cazul task-urilor TR stricte, depăşirea acestor termene implică funcţionarea eronată a sistemului, uneori cu consecinţe catastrofale pentru operatori şi pentru mediul înconjurător.

Definiția 4-7.

Definim *termenul limită* al task-ului M_i (*deadline*) pentru ciclul k de execuție, notat cu $t_{dl,k}^{M_i}$, instanța de timp până la care execuția lui M_i are valoare în cadrul sistemului TR.

❑

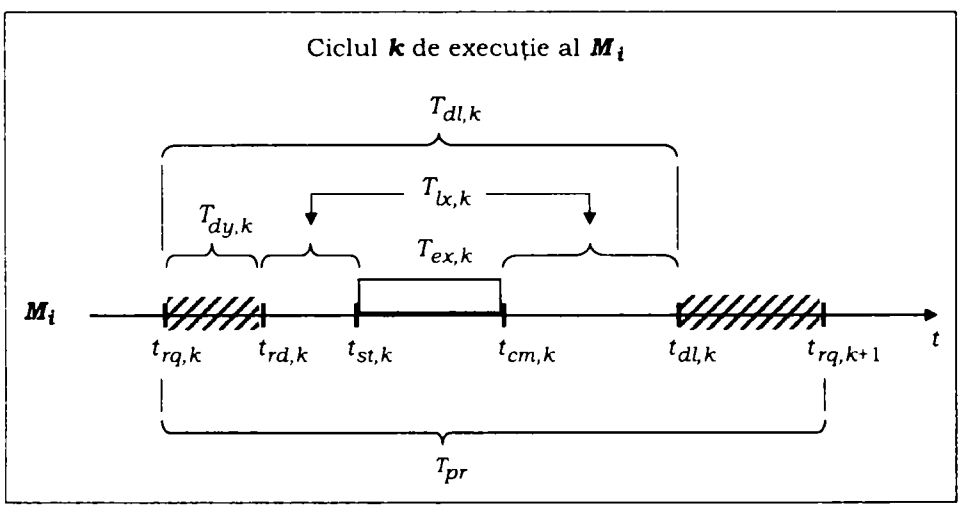

Figura 4-3. Modelul temporal al task-ului TR strict, M_i

Figura 4-3 şi Tabelul 4-3 de mai jos sintetizează modelul şi proprietățile temporale discutate până în acest punct, ce caracterizează task-ul TR (strict) M_i în cadrul unui ciclu de execuție, k. Intervalele de timp haşurate în figură semnifică faptul că execuția lui M_i nu este posibilă: pe de o parte, în intervalul $[t_{rq,k}^{M_i}, t_{rd,k}^{M_i}]$ task-ul M_i nu este gata pentru execuție (vezi Definiția 4-6), iar pe de altă parte, în intervalul $[t_{dl,k}^{M_i}, t_{rq,k+1}^{M_i}]$ M_i îşi depăşeşte termenul de execuție, implicând operarea eronată a sistemului.

Între parametrii temporali ai modelului de task TR, prezentați în Tabelul 4-3, există următoarele relații:

$$T_{ex,k}^{M_i} = t_{cm,k}^{M_i} - t_{st,k}^{M_i} \tag{4-9}$$

$$T_{dl,k}^{M_i} = t_{dl,k}^{M_i} - t_{rq,k}^{M_i} \tag{4-10}$$

$$T_{dy,k}^{M_i} = t_{rd,k}^{M_i} - t_{rq,k}^{M_i} \tag{4-11}$$

$$T_{lx,k}^{M_i} = T_{dl,k}^{M_i} - T_{dy,k}^{M_i} - T_{ex,k}^{M_i} \tag{4-12}$$

şi relația (4-6):

$$T_{pr}^{M_i} = \min_{1 \le k \le N^{M_i}} \left\{ \left(t_{rq,k+1}^{M_i} - t_{rq,k}^{M_i} \right) \right\}.$$

Tabelul 4-3. Parametrii temporali ai task-ului strict M_i

Parametru	Denumire	Semnificaţie
$t_{rq,k}^{M_i}$	*Momentul de invocare (Request time)*	Momentul în care M_i este invocat de STR. Defineşte începutul ciclului k de execuţie a lui M_i.
$t_{rd,k}^{M_i}$	*Momentul gata de execuţie (Ready time)*	Momentul de la care M_i poate fi planificat pentru execuţie.
$t_{st,k}^{M_i}$	*Momentul de start (Start time)*	Momentul în care M_i îşi începe efectiv execuţia.
$t_{cm,k}^{M_i}$	*Momentul de terminare (Completion time)*	Momentul în care M_i îşi termină execuţia şi este eliberat procesorul sistemului.
$t_{dl,k}^{M_i}$	*Termenul limită (Deadline)*	Momentul până la care execuţia lui M_i are valoare în cadrul sistemului.
$T_{ex,k}^{M_i}$	*Durata de execuţie (Execution interval)*	Intervalul de timp necesar execuţiei lui M_i în cadrul sistemului.
$T_{dl,k}^{M_i}$	*Intervalul limită (Deadline interval)*	Intervalul de timp în care execuţia lui M_i poate fi planificată pentru o funcţionare corectă a sistemului.
$T_{dy,k}^{M_i}$	*Intervalul de întârziere (Delay interval)*	Intervalul de timp după care execuţia lui M_i poate fi planificată.
$T_{lx,k}^{M_i}$	*Intervalul disponibil planificării (Laxity interval)*	Intervalul de timp disponibil pentru planificarea lui M_i astfel încât execuţia lui să nu depăşească termenul limită.
$T_{pr}^{M_i}$	*Perioada (Period)*	Durata minimă a unui ciclu de execuţie a lui M_i.
N^{M_i}	*Numărul de execuţii (Execution count)*	Numărul total de execuţii ale lui M_i (numărul de cicluri de execuţie).

Pentru ca planificarea task-ului M_i să poată fi fezabilă, adică să existe posibilitatea planificării şi execuţiei lui M_i în cadrul sistemului TR, parametrii săi temporali vor trebui să verifice relaţiile:

$$t_{rq,k}^{M_i} \le t_{rd,k}^{M_i} \le t_{st,k}^{M_i} < t_{cm,k}^{M_i} \le t_{dl,k}^{M_i} \le t_{rq,k+1}^{M_i} \tag{4-13}$$

sau, echivalent:

$$0 < T_{ex,k}^{M_i} \le T_{dl,k}^{M_i} \le T_{pr}^{M_i} \tag{4-14}$$

$$0 \le T_{dy,k}^{M_i} \le T_{dl,k}^{M_i} - T_{ex,k}^{M_i} < T_{dl,k}^{M_i} \le T_{pr}^{M_i} \tag{4-15}$$

pentru $\forall\, k \in \{1, \ldots, N^{M_i}\}$.

Relaţiile (4-9) până la (4-15) pot fi utilizate ca un prim pas în faza de validare a specificaţiilor unui STR din punct de vedere al comportării în domeniul timp.

Urmând clasificarea semnalelor, detaliată în subcapitolul precedent, vom trata în continuare aspectele legate de comportamentul în timp al task-urilor unui sistem TR, vis a vis de cel al semnalelor, vizând în special cazul task-urilor TR stricte.

Semnalele de ieşire

Semnalele de ieşire ale unui STR sunt *generate* de către task-uri ale sistemului. Aici, relaţia de cauzalitate este de tipul "acţiune → eveniment". Comportamentul în domeniul timp al semnalelor de ieşire este complet determinat de proprietăţile temporale ale task-urilor care le generează. Prin urmare, specificaţiile de comportare temporală ale unui semnal de ieşire sunt direct determinate de cele ale task-ului care îl generează. De exemplu, un task M_i care este periodic, de perioadă $T_{pr}^{M_i}$, şi care are un număr de N^{M_i} execuţii, va genera un semnal de ieşire S_j periodic şi temporar, de perioadă $T_{pr}^{S_j} = T_{pr}^{M_i}$ şi cu un număr $N^{S_j} = N^{M_i}$ de apariţii.

În faza de specificare/verificare a STR este necesară punerea în corespondenţă a parametrilor temporali ai semnalelor de ieşire cu parametrii temporali ai task-urilor care le generează.

Semnalele interne

Semnalele interne al unui STR sunt, la rândul lor, *generate* de către task-uri ale sistemului (fiind deci echivalente cu semnalele de ieşire), iar pentru alte task-uri, reprezintă semnale de intrare. Problema sincronizării şi comunicaţiei dintre task-urile STR va fi tratată separat, în paragrafele ce descriu modelul task-ului TR strict ("ModX").

Semnalele de intrare

Despre semnalele de intrare ale unui STR, spunem că sunt *tratate* de către task-uri ale sistemului. Relaţia de cauzalitate în acest caz este de tipul "eveniment → acţiune", comportamentul în timp al task-urilor ce tratează semnalele de intrare trebuind să corespundă specificaţiilor temporale ale semnalelor.

Perspectiva *tratării* semnalelor de intrare de către task-uri ale STR impune adăugarea a încă unui parametru temporal important la modelul de semnal propus în subcapitolul anterior (vezi Tabelul 4-2). Este vorba despre aşa-numitul *interval de răspuns* al sistemului (*response time*), $T_{rs}^{S_i}$, la un anumit semnal S_i. Echivalent, timpul de răspuns se referă la task-ul care tratează semnalul respectiv.

Definiţia 4-8.

Definim *intervalul de răspuns* pentru semnalul S_i, notat cu $T_{rs}^{S_i}$, intervalul de timp iniţiat de momentul apariţiei lui S_i şi în decursul căruia task-ul corespunzător al sistemului trebuie să trateze pe S_i.

❑

Tratarea semnalelor de intrare se poate realiza prin două tehnici de bază, diferite: în manieră asincronă, utilizând mecanismul de întreruperi al STR, sau în manieră sincronă, prin baleierea continuă, periodică a stării semnalului de intrare (tehnica denumită *polling*). Mecanismul de întreruperi şi tratarea semnalelor de intrare în manieră asincronă presupune planificarea şi execuţia de task-uri aperiodice, invocate de întreruperi. Existenţa în cadrul STR a acestor task-uri afectează puternic predictibilitatea şi funcţionarea deterministă a sistemului (discuţia din Capitolul 3).

Aşadar, pentru cazul sistemelor TR stricte, este necesară eliminarea task-urilor aperiodice şi utilizarea în exclusivitate a celor periodice. Această cerinţă este realizabilă din punct de vedere practic, din următoarele considerente:

- Chiar dacă o anume aplicaţie TR utilizează task-uri aperiodice, există mecanisme demonstrate în literatura de specialitate, care permit transformarea lor în task-uri periodice [Mok 83, Mok 84].
- Task-urile periodice permit analiza operării sistemului TR într-o manieră predictibilă, deterministă şi conceperea unor algoritmi de planificare fezabili, care să respecte specificaţiile temporale de execuţie ale task-urilor TR stricte.
- Utilizarea mecanismului de polling împreună cu mecanisme de stocare tampon (registre tampon, buffere, zone de memorie tampon) pentru tratarea semnalelor de intrare cu ajutorul task-urilor periodice este o soluţie fezabilă.
- Două din dezavantajele majore ale acestui tip de abordare constau în scăderea drastică a eficienţei sistemului şi lipsa acută de flexibilitate. Mecanismele de întreruperi au fost concepute tocmai pentru tratarea cât mai eficient posibil a evenimentelor asincrone de către sistem. Problema constă însă în lipsa de predictibilitate a sistemului implicată de utilizarea întreruperilor [Stewart 01].

Teorema 4-1. Tratarea semnalelor de intrare cu task-uri periodice simple

Considerăm un task TR strict, periodic, simplu, M_i, caracterizat prin următorii parametri temporali:

- $T_{pr}^{M_i}$, perioada lui M_i,
- $T_{ex}^{M_i}$, durata de execuţie a lui M_i: $0 < T_{ex}^{M_i} \leq T_{pr}^{M_i}$,
- Intervalul limită al lui M_i este egal cu perioada acestuia: $T_{dl}^{M_i} = T_{pr}^{M_i}$,
- M_i este gata de execuţie la începutul fiecărei perioade: $T_{dy}^{M_i} = 0$.

Dacă M_i tratează un semnal de intrare S_j într-un interval de răspuns $T_{rs}^{S_j}$, atunci perioada lui M_i trebuie să respecte relaţia[1]:

$$T_{pr}^{M_i} \leq \left\lfloor \frac{T_{rs}^{S_j} + 1}{2} \right\rfloor \tag{4-16}$$

[1] Cu $\lfloor x \rfloor$ ("floor") s-a notat cel mai mare întreg mai mic sau egal cu x, iar cu $\lceil x \rceil$ ("ceiling"), cel mai mic întreg mai mare sau egal cu x.

Demonstraţie

Vom demonstra teorema luând în considerare cazul cel mai defavorabil care poate să apară relativ la momentul începerii execuţiei lui M_i faţă de momentul apariţiei lui S_j: semnalul apare în instanţa imediat următoare începerii execuţiei din ciclul curent a lui M_i, iar intervalul de timp dintre două execuţii consecutive este maxim (vezi Figura 4-4).

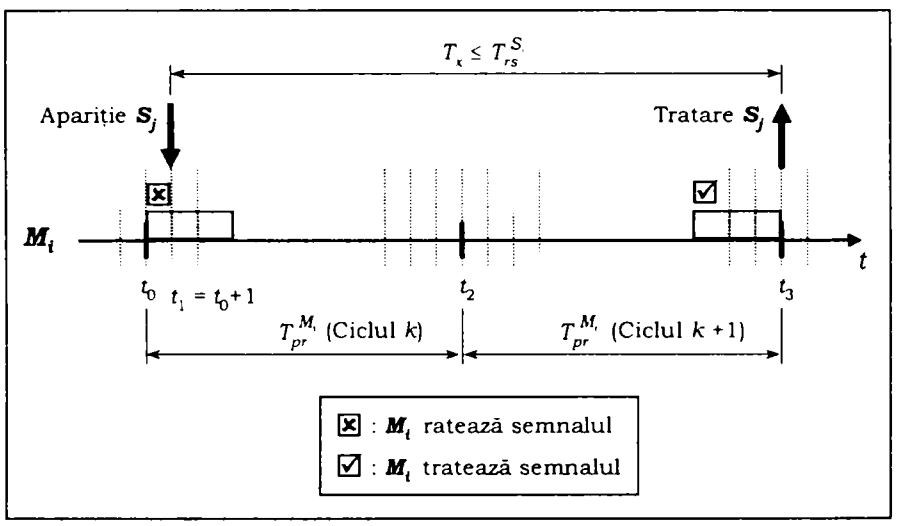

Figura 4-4. Cazul cel mai defavorabil tratării semnalului S_j cu un task M_i

Din figura de mai sus se observă că execuţia k a lui M_i ratează apariţia lui S_j, starea acestuia (informaţia furnizată de semnal) fiind reţinută într-o structură tampon a sistemului, aşa cum am descris mai sus, pentru tratarea în cadrul unei execuţii ulterioare a lui M_i. În situaţia cea mai defavorabilă, execuţia $(k + 1)$ a lui M_i are loc la sfârşitul ciclului de execuţie următor (la sfârşitul perioadei următoare), pe când execuţia curentă are loc la începutul perioadei.

Intervalul de timp scurs de la apariţia semnalului S_j şi până M_i finalizează tratarea acestuia are valoarea:

$$T_x = 2 \cdot T_{pr}^{M_i} - 1 \tag{4-17}$$

Dar, pentru ca S_j să fie tratat în intervalul de răspuns specificat, trebuie ca $T_x \le T_{rs}^{S_j}$, de unde, prin înlocuirea lui T_x în (4-17), obţinem relaţia (4-16), cerută a fi demonstrată în enunţul teoremei.

❑

Teorema 4-1 este utilă în fazele de specificare, verificare şi analiză a STR stricte, pentru cazurile în care este necesară deducerea (automată) a comportamentului unor task-uri care tratează semnale de intrare, în condiţiile în care se specifică doar parametrii temporali ai semnalelor. Din punctul de vedere al programatorului de aplicaţii, este mai utilă şi mai intuitivă specificarea directă a

comportării temporale ale semnalelor de intrare în sistem decât deducerea manuală a caracteristicilor task-urilor TR stricte care vor trebui să trateze aceste semnale.

În continuare exemplificăm cu câteva cazuri concrete specificarea temporală a unui semnal de intrare şi aplicarea Teoremei 4-1 pentru deducerea parametrilor task-ului TR strict care tratează semnalul.

Exemplul 4-2.

Considerăm un STR cu două semnale de intrare, S_1 şi S_2. Pentru tratarea semnalului S_1 cu task-ul M_1 se specifică un interval de răspuns $T_{rs}^{S_1} = 11$, iar pentru tratarea lui S_2 cu M_2, intervalul de răspuns este $T_{rs}^{S_2} = 10$. Durata de execuţie a ambelor task-uri o considerăm $T_{ex}^{M_1} = T_{ex}^{M_2} = 3$ unităţi de timp.

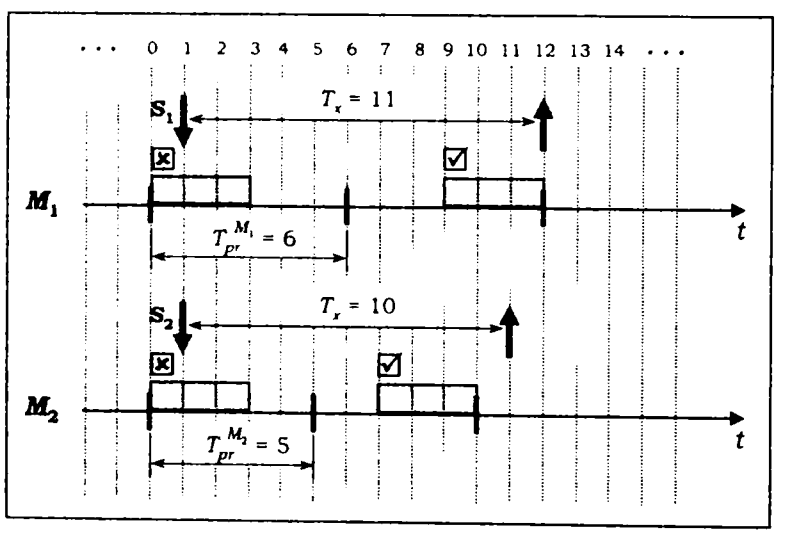

Figura 4-5. Task-urile M_1 şi M_2 ce tratează semnalele S_1, respectiv S_2

Utilizând Teorema 4-1 pentru determinarea parametrilor temporali ai lui M_1 şi M_2 (a perioadelor acestora), rezultă (vezi Figura 4-5):

$$T_{pr}^{M_1} \le \left\lfloor \frac{T_{rs}^{S_1} + 1}{2} \right\rfloor = \left\lfloor \frac{11+1}{2} \right\rfloor = 6,$$

pentru M_1, şi:

$$T_{pr}^{M_2} \le \left\lfloor \frac{T_{rs}^{S_2} + 1}{2} \right\rfloor = \left\lfloor \frac{10+1}{2} \right\rfloor = 5.$$

❑

Teorema 4-1 permite tratarea cu ajutorul task-urilor TR stricte periodice a semnalelor aperiodice de intrare pentru care se specifică intervalul de răspuns.

În cazurile în care, pentru un anumit semnal aperiodic de intrare S_j nu se specifică intervalul de răspuns $T_{rs}^{S_j}$, ci perioada $T_{pr}^{S_j}$ (intervalul de timp minim dintre oricare două apariţii consecutive), se poate considera că timpul de răspuns al task-ului M_i care tratează pe S_j este determinat de această perioadă. Cu alte cuvinte, M_i trebuie să trateze pe S_j până când are loc o nouă apariţie a acestuia:

$$T_{pr}^{M_i} \leq \left\lfloor \frac{T_{pr}^{S_j} + 1}{2} \right\rfloor . \tag{4-18}$$

Pentru exemplificare, pot fi considerate cazurile reprezentate în Figura 4-4 şi Figura 4-5, cu $T_x = T_{pr}^{S_j}$.

Dacă sunt specificaţi ambii parametri temporali ai unui semnal aperiodic de intrare S_j, $T_{pr}^{S_j}$ şi $T_{rs}^{S_j}$, trebuie întâi verificată în mod obligatoriu relaţia

$$T_{rs}^{S_j} \leq T_{pr}^{S_j} \tag{4-19}$$

care afirmă că, practic, intervalul alocat tratării lui S_j de către STR nu poate depăşi ca durată intervalul celei mai apropiate apariţii consecutive a lui S_j. Dacă inegalitatea este confirmată, în formula de determinare a perioadei task-ului M_i rămâne parametrul $T_{rs}^{S_j}$ ca în (4-16).

Formula (4-18) poate fi aplicată şi pentru tratarea semnalelor periodice de intrare, fiind o procedură sigură, care evită "scăparea" vreunei apariţii a semnalului: în cadrul unei perioade a lui S_j, $T_{pr}^{S_j}$, task-ul TR strict M_i este planificat şi executat de două ori. Dacă într-o perioadă a semnalului, acesta apare înaintea execuţiei lui M_i, S_j este tratat în cel de-al doilea ciclu de execuţie, înainte unei noi apariţii. Problema acestei abordări însă constă în scăderea drastică a eficienţei sistemului – preţul plătit de sistemele TR stricte în schimbul garantării îndeplinirii tuturor specificaţiilor temporale ale aplicaţiei, în orice condiţii de operare.

Exemplul 4-3.

Considerăm un STR cu un semnal periodic de intrare, S_1, cu perioada $T_{pr}^{S_1} = 7$, de tratarea căruia este responsabil task-ul TR strict şi periodic, M_1. Durata de execuţie a lui M_1 o considerăm constantă pentru toate ciclurile de execuţie: $T_{ex}^{M_1} = 2$.

Din Teorema 4-1 rezultă o perioadă $T_{pr}^{M_1} \leq \left\lfloor \frac{T_{pr}^{S_1} + 1}{2} \right\rfloor = \left\lfloor \frac{7+1}{2} \right\rfloor = 4$ pentru M_1. O configuraţie posibilă de operare a sistemului este reprezentată grafic în Figura 4-6, iar o scurtă evaluare a eficienţei execuţiei lui M_1 în cadrul sistemului este redată în Tabelul 4-4.

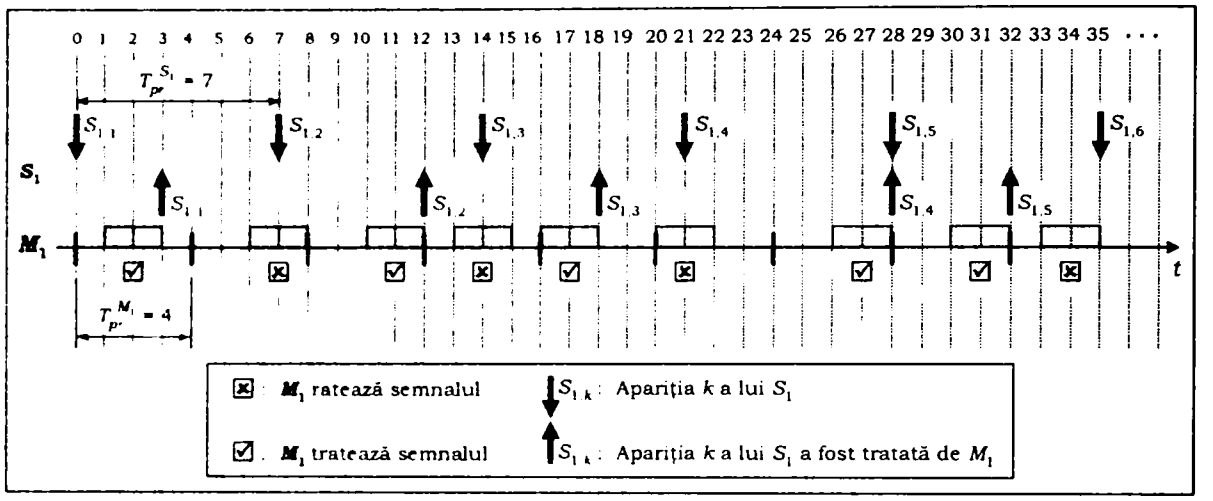

Figura 4-6. Exemplificare a tratării unui semnal de intrare periodic

Tabelul 4-4. Eficienţa task-ului TR strict M_1

Total apariţii S_1	6
Total execuţii M_1	9
Execuţii M_1 care tratează S_1	5
Execuţii M_1 care ratează S_1	4
Eficienţă execuţie M_1	55 %

❑

Pentru creşterea eficienţei tratării semnalelor de intrare periodice, se pot utiliza task-uri TR periodice, cu aceeaşi perioadă ca a semnalului de intrare, cu condiţia sincronizării apariţiei semnalului cu execuţia task-ului. Problema sincronizării dintre un task periodic şi un semnal de intrare de aceeaşi perioadă, este relativ dificil de rezolvat, cu atât mai mult cu cât în foarte multe aplicaţii practice, sistemul TR şi dispozitivul extern care generează semnalul utilizează oscilatoare distincte. Astfel, oricât de bine ar fi specificată perioada pentru semnalul de intrare şi pentru task-ul TR strict care îl tratează, după un anumit interval de timp de operare, apare un defazaj sensibil între apariţia semnalului şi execuţia task-ului, rezultând în final la pierderea unei perioade.

O variantă de rezolvare a problemei sincronizării este ca sistemul să se comporte ca un "master" pentru dispozitivul extern care generează semnalul de intrare periodic: fiecare apariţie a semnalului să fie declanşată de fapt tot de STR (de un task al sistemului), sub forma activării periodice a unei linii de semnal sau a unei comenzi către dispozitivul extern, iar timpul de răspuns al echipamentului să fie cunoscut.

Observaţie

Tratarea semnalelor de intrare periodice şi aperiodice mai poate fi rezolvată în cadrul STR şi prin implementarea unui mecanism de tipul *server periodic de întreruperi* (*sporadic server, aperiodic server, deferrable server*) [Lehoczky 87, Sprunt 89a, Sprunt 89b], care, în principiu, constă dintr-un task periodic ce se

ocupă în cadrul fiecărei execuţii de arbitrarea şi tratarea, una câte una, a întreruperilor ce apar în sistem.

Această abordare are, la rândul ei, două inconveniente majore:

- durata de execuţie a serverului de întreruperi va trebui să fie cel puţin egală cu cea mai mare durată de execuţie corespunzătoare tratării oricărui semnal aperiodic din cele gestionate de server,
- perioada serverului va trebui să fie suficient de mică pentru a putea satisface specificaţiile temporale ale tuturor semnalelor gestionate, din punct de vedere al intervalului de răspuns şi/sau al perioadei acestora.

Cu cât sunt mai multe semnale de intrare (evenimente asincrone) tratate cu mecanismul serverului de întreruperi şi cu cât acestea au specificaţii temporale mai diferite între ele, cu atât mai dificilă este planificarea şi execuţia periodică a serverului.

4.4 Modelul task-ului TR strict: ModX-ul

Elementele legate de gestionarea timpului, a semnalelor şi a task-urilor într-un STR strict discutate până acum în cadrul capitolului de faţă, cât şi problemele curente de proiectare şi implementare a STR stricte prezentate în Capitolul 3, au ca rezultat propunerea unui model de task TR strict – *ModX*-ul.

Un *ModX* (*Modul executabil*) reprezintă un task TR periodic, modular, cu specificaţii complete şi stricte de comportare în timp şi cu planificare şi execuţie în context non-preemptiv, destinat a servi ca element de bază în specificarea, proiectarea şi dezvoltarea sistemelor şi aplicaţiilor TR stricte, pentru care determinismul şi predictibilitatea operării sunt cerinţe esenţiale.

Din perspectiva arhitecturii aplicaţiilor TR stricte, cât şi din punct de vedere funcţional, ModX-ul este un modul software, adică echivalentul unui "bloc de bază" (*basic block*) utilizat în teoria analizei programelor şi a compilarii [Acho 87, Muchnick 97]. ModX-ul prezintă un set de intrări (parametri de intrare), un corp (bloc) de instrucţiuni care prelucrează intrările, variabilele definite local şi variabilele globale, şi un set de ieşiri (parametri de ieşire). Din punct de vedere al specificaţiilor şi al comportamentului în domeniul timp, parametrii de intrare/ieşire ai ModX-ului sunt semnale interne ale sistemului. În plus, un ModX poate interacţiona cu semnale de intrare (ModX-ul *tratează* semnale de intrare) şi de ieşire (ModX-ul *generează* semnale de ieşire).

Formal, ModX-ul este o componentă a unei aplicaţii TR reprezentate sub forma unui graf de program, direcţionat, aciclic [Ferrante 87, Podgurski 89, Marlowe 90, Niehaus 91, Gupta 96]:

$$\boldsymbol{G} \equiv \langle \boldsymbol{M}, \boldsymbol{V} \rangle \tag{4-20}$$

unde $\boldsymbol{M} = \{M_i \mid 1 \le i \le |\boldsymbol{M}|\}$ este mulţimea nodurilor grafului $\boldsymbol{G}$ (ModX-urile aplicaţiei) şi $\boldsymbol{V} = \{(M_i, M_j) \mid M_i, M_j \in \boldsymbol{M}\ ;\ 1 \le i, j \le |\boldsymbol{M}|\}$ este mulţimea arcelor lui $\boldsymbol{G}$, constând din perechi ordonate de noduri.

Definiţia 4-9.

Un ModX M_i este un task TR strict, reprezentat formal prin următorul model:

$$M_i \equiv \langle \boldsymbol{T}, \boldsymbol{P}, \boldsymbol{S}, \boldsymbol{F} \rangle \tag{4-21}$$

unde:

$\boldsymbol{T} = \{ T_{pr}^{M_i}, T_{ex}^{M_i}, T_{dl}^{M_i}, T_{dy}^{M_i}, N^{M_i} \}$ este setul specificaţiilor de comportare temporală a ModX-ului M_i,

$\boldsymbol{P} = \{\boldsymbol{P_{IN}}, \boldsymbol{P_{OUT}}, \boldsymbol{P_{GLB}}\}$ este setul parametrilor de intrare, de ieşire, respectiv al variabilelor globale cu care operează M_i,

$\boldsymbol{S} = \{\boldsymbol{S_{IN}}, \boldsymbol{S_{OUT}}\}$ este setul semnalelor de intrare tratate de M_i şi al semnalelor de ieşire generate de M_i, iar

$\boldsymbol{F}$ este setul de instrucţiuni care procesează informaţia în cadrul M_i, reprezentând specificarea funcţională a ModX-ului.

❑

În continuare vom aborda pe rând fiecare componentă a strucurii modelului de task TR stric propus – ModX-ul.

Task-urile periodice, non-preemptive şi predictibilitatea STR stricte

Pentru a se putea obţine un sistem TR care să opereze în condiţii maxime de predictibilitate şi determinism, şi ale cărui task-uri să poată fi executate cu respectarea în orice condiţii a termenelor impuse, este necesară eliminarea elementelor asincrone, sporadice din sistem. Principalul mecanism care generează o comportare impredictibilă, asincronă pentru sistemele digitale îl reprezintă întreruperile [Stewart 01].

Ca rezultat, ModX-ul a fost conceput ca un task periodic, planificat şi executat în cadrul STR în regim non-preemptiv. Aceste două caracteristici principale ale ModX-ului facilitează specificarea şi verificarea comportamentului temporal al aplicaţiei, cât şi analiza fezabilităţii planificării task-urilor înaintea încărcării şi execuţiei lor efective în sisteme TR a căror fiabilitate de operare este critică. Un alt avantaj constă în eliminarea problemelor de tip inversare a priorităţii, tipice sistemelor TR care implementează tehnici de planificare şi execuţie preemptive, bazate pe priorităţi.

Abordarea modulară

Pentru concepţia modelului de task TR strict a fost aleasă o abordare modulară din mai multe considerente:

- Creşterea flexibilităţii în procesul de dezvoltare a aplicaţiilor TR. Dezvoltarea modulară şi vizuală a aplicaţiilor, utilizând reprezentări de tip graf, uşurează munca de proiectare, specificare, validare şi analiză a acestora.
- Evidenţierea rapidă a dependenţelor de program (control şi date) ale task-urilor aplicaţiei.

- Rezolvarea simplă a comunicaţiei şi sincronizării între task-urile aplicaţiei. Există două mecanisme puse la dispoziţia programatorului de aplicaţii: transmiterea de parametri de intrare/ieşire şi utilizarea (limitată) a variabilelor globale.
- Ţinând cont de faptul că task-urile aplicaţiei sunt periodice, este mai uşoară şi mai intuitivă vizualizarea execuţiei acestora ca module în cadrul aplicaţiei.

Aspectul funcţional şi evaluarea duratei de execuţie

Din punct de vedere funcţional, ModX-urile reprezintă secvenţe de cod ce se execută fără blocare şi în regim non-preemptiv. Prin urmare, operaţiile implementate de un ModX sunt *operaţii atomice*, eliminându-se astfel necesitatea mecanismelor de sincronizare şi control al acceselor concurente la resurse.

O problemă extrem de importantă pentru studiul STR stricte este determinarea duratei de execuţie a task-urilor. Parametrul temporal $T_{ex,k}^{M_i}$ specifică durata de execuţie a task-ului M_i, care variază de la un ciclu k de execuţie la altul (vezi Definiţia 4-2 şi Tabelul 4-3). Pentru a se putea dezvolta algoritmi fezabili de planificare, este necesară considerarea pentru fiecare task a unei durate de execuţie constante, $T_{ex}^{M_i}$, care să nu varieze de la un ciclu de execuţie la altul. Pe de altă parte, planificarea task-urilor unui STR strict trebuie să fie fezabilă pentru cele mai dezavantajoase situaţii de operare ale sistemului şi nu doar pentru cazurile de încărcare medie, de exemplu. Din aceste considerente, durata de execuţie a unui task TR strict, pentru fiecare ciclu de execuţie, este considerată ca fiind egală cu *WCET* (*Worst Case Execution Time*), adică durata de execuţie cea mai defavorabilă [Puschner 89, Puschner 91, Park 91, Burns 91, Chapman 94, Parashkevov 94, Puschner 00]. Durata *WCET* pentru un task oarecare se determină ca fiind suma timpilor de execuţie ai instrucţiunilor ce compun calea cea mai lungă de execuţie a task-ului şi considerând condiţiile cele mai defavorabile de operare.

WCET nu poate fi determinat fără o serie de restricţii impuse limbajului de programare cu ajutorul căruia se specifică comportamentul funcţional al task-urilor unui STR [Shaw 89, Stoyenko 91, Chung 95, Gerber 97]. Din aceste motive, o serie de limbaje de programare au fost modificate (adăugându-se restricţii, etc.) şi au fost concepute şi dezvoltate noi limbaje de programare, vizând facilitarea analizei programelor TR şi a determinării *WCET*: Real-Time EUCLID [Klingerman 86], CRL [Stoyenko 95], etc.

Programarea aplicaţiilor TR stricte cu ajutorul ModX-urilor este supusă, la rândul ei, următoarelor restricţii principale:

- Utilizarea *recursivităţii* – este interzisă, din cauza cerinţelor de spaţiu impredictibile pe care le produce (dimensiunea memoriei necesare).
- *Buclele de program* trebuie să fie limitate ca durată totală şi ca spaţiu. Cu alte cuvinte, trebuie ca numărul total de iteraţii ale fiecărei bucle să fie bine determinat. De asemenea, buclele încuibate (*nested loops*) se vor supune aceloraşi restricţii, inclusiv unei limitări a nivelurilor de încuibare, pentru o analiză mai uşoară a programului.

- Utilizarea *instrucţiunilor de ramificare* de tip `GOTO` sunt permise doar în condiţiile în care au ca efect salturi de tip "înainte" ale execuţiei. În acest fel se păstrează reductibilitatea grafului dependenţelor de control ale programului [Acho 86, Muchnick 97] şi se permite reprezentarea programului sub formă de graf aciclic. Salturile de tip "înapoi" sunt permise doar în structuri de tip buclă de program (de exemplu: `FOR`, `WHILE`, `REPEAT`).
- Utilizarea *apelurilor de proceduri şi funcţii* în cadrul ModX-urilor este permisă dar într-o manieră limitată şi liniară. Trebuie subliniat că, prin arhitectura modulară de programare propusă prin intermediul ModX-urilor, funcţiile şi procedurile pot fi implementate ca ModX-uri individuale. Acest lucru este avantajos şi din perspectiva planificării, având în vedere că ModX-urile se execută în context non-preemptiv. Cu cât un ModX particular durează mai mult ca execuţie, cu atât va fi mai dificil de planificat în cadrul sistemului TR.
- *Alocarea dinamică a resurselor* (de exemplu, a memoriei) – este interzisă, pentru că mecanismele curente de alocare dinamică prezintă un comportament impredictibil în timp. Pentru a facilita analiza "offline" a sistemului, mecanismele utilizate trebuie să fie statice şi liniare.

Limbajele pe care le avem în vedere pentru programarea ModX-urilor trebuie să se apropie cât mai mult de limbajul de asamblare al procesorului ţintă şi să respecte restricţiile enumerate mai sus. Astfel, într-o primă fază se utilizează direct limbajul de asamblare [Micea 04b], şi intenţionăm adaptarea unei versiuni de "ANSI C" pentru programarea aplicaţiilor TR. Compilarea/asamblarea fiecărui ModX în parte se face cu opţiuni de generare de cod relocabil (bibliotecă de module).

Pentru fiecare ModX M_i din cadrul unei aplicaţii TR, se va considera o durată de execuţie constantă, $T_{ex}^{M_i}$, egală cu *WCET* al codului ModX-ului M_i (incluzând apelurile interne la proceduri şi funcţii).

Parametrii de intrare/ieşire ai ModX-ului şi comunicaţia inter-task

Comunicaţia între ModX-urile aplicaţiei TR se realizează prin intermediul setului de parametri de intrare, $\boldsymbol{P_{IN}}$, şi al celor de ieşire, $\boldsymbol{P_{OUT}}$, ale fiecărui ModX. Un ModX M_i nu poate începe ciclul curent de execuţie până când toţi parametrii săi de intrare nu au fost actualizaţi de ModX-urile care au aceiaşi parametri ca parametri de ieşire.

$\boldsymbol{P_{GLB}}$ reprezintă setul acelor variabile globale ale aplicaţiei care sunt accesate de către un ModX. Variabilele globale reprezintă un al doilea mecanism de sincronizare şi comunicare inter-task pus la dispoziţia programatorului de aplicaţie. Datorită faptului că ModX-urile se execută în context non-preemptiv şi, astfel, implementează operaţii atomice, nu sunt necesare mecanisme suplimentare de control şi sincronizare a acceselor concurente la variabilele globale ale aplicaţiei. Se recomandă, totuşi, utilizarea cu atenţie a variabilelor globale pentru că ele practic ascund dependenţa normală de date dintre task-urile aplicaţiei, cu implicaţii în construirea unui graf al dependenţelor de program realist. O situaţie în care se recomandă utilizarea variabilelor globale este aceea în care aplicaţia foloseşte structuri de date de dimensiuni mari (cum sunt de exemplu, diverse buffere de semnale sau cadre de

imagini). Prin această tehnică, se evită copierea repetată a structurilor de mari dimensiuni de la o execuţie la alta a ModX-urilor, sub formă de parametri de intrare/ieşire.

Observaţie

Un alt aspect important ce rezultă din faptul că ModX-urile sunt task-uri periodice este posibilitatea transmiterii unei anumite stări curente a unui ModX, de la o execuţie a sa, la alta. Această operaţie este posibilă prin stabilirea unui parametru (sau a mai multora), atât ca parametru de intrare, cât şi de ieşire. În acest context însă, este necesară implementarea unui task de iniţializare a aplicaţiei (incluzând setarea valorilor iniţiale pentru astfel de parametri), care să se execute primul, înaintea celorlaltor ModX-uri. Acesta are un regim special de execuţie, nefiind periodic ci executându-se o singură dată.

Tratarea şi generarea semnalelor

ModX-urile interacţionează cu semnalele de intrare/ieşire ale STR în maniera descrisă în secţiunea precedentă. Din punct de vedere al programatorului de aplicaţii, setul de semnale $\boldsymbol{S} = \{\boldsymbol{S_{IN}}, \boldsymbol{S_{OUT}}\}$ cu care interacţionează un ModX M_i este perceput la nivelul specificării comportamentului temporal al semnalelor. Informaţia pe care o poartă un semnal oarecare este manipulată de STR prin intermediul variabilelor locale, globale şi al parametrilor de intrare/ieşire a ModX-urilor. Astfel, pentru modelul propus, un semnal S_j este reprezentat de parametrii săi temporali:

$$S_j \equiv \{ T_{pr}^{S_j}, T_{rs}^{S_j}, T_{act}^{S_j}, N^{S_j} \}, \text{cu } (S_j \in \boldsymbol{S_{IN}}) \wedge (S_j \in \boldsymbol{S_{OUT}}) \qquad (4\text{-}22)$$

unde: $T_{pr}^{S_j}$ este perioada semnalului S_j (dacă e periodic) sau durata minimă de timp ce separă oricare două apariţii consecutive ale acestuia, $T_{act}^{S_j}$ este durata intervalului în care au loc apariţii ale semnalului S_j, şi N^{S_j} este numărul de apariţii ale lui S_j (vezi Tabelul 4-2), iar, dacă S_j este semnal de intrare, $T_{rs}^{S_j}$ reprezintă intervalul de timp iniţiat de momentul apariţiei lui S_j şi în decursul căruia M_i trebuie să-l trateze (vezi Definiţia 4-8).

Specificarea comportamentului temporal al ModX-urilor

Parametrii temporali ai ModX-ului M_i aparţin mulţimii $\boldsymbol{T} = \{ T_{pr}^{M_i}, T_{ex}^{M_i}, T_{dl}^{M_i}, T_{dy}^{M_i}, N^{M_i} \}$, unde:

- $T_{pr}^{M_i}$ este perioada lui M_i – caracteristică esenţială şi obligatorie pentru toate ModX-urile, fiind task-uri periodice. Perioada este constantă de-a lungul operării aplicaţiei, deci nu depinde de ciclul curent k de execuţie a lui M_i ;

- $T_{ex}^{M_i}$ este durata de execuţie a lui M_i. Este de asemenea constantă pe durata operării sistemului TR, fiind egală cu *WCET* al codului ModX-ului;
- $T_{dl}^{M_i}$ este intervalul limită pentru planificarea şi execuţia lui M_i în cadrul perioadei acestuia. $T_{dl}^{M_i}$ este un parametru opţional, introdus pentru cazurile în care este necesar ca execuţia lui M_i să se finalizeze cu un interval de timp oarecare înainte de terminarea perioadei, pentru toate ciclurile de execuţie;
- $T_{dv}^{M_i}$ este intervalul de întârziere a planificării şi execuţiei lui M_i în cadrul perioadei. $T_{dv}^{M_i}$ este, la rândul lui un parametru opţional, ce poate fi utilizat în cazurile în care se doreşte ca planificarea şi execuţia lui M_i să nu poată fi efectuate într-un anume interval de la începutul fiecărei perioade a lui M_i.
- N^{M_i} este numărul total de execuţii ale lui M_i. Este un parametru opţional, care tratează trei cazuri:
  - M_i are un număr nedefinit de execuţii (execuţie continuă): $N^{M_i} = \infty$,
  - M_i are un număr finit, bine precizat de execuţii: $N^{M_i} < \infty$. În acest caz, M_i interacţionează cu semnale (generează sau tratează semnale) cu apariţie temporară,
  - M_i nu mai este executat de către sistem: $N^{M_i} = 0$. M_i este planificat în continuare, dar nu va mai fi lansat în execuţie de către executivul/dispatcher-ul sistemului.

Observaţie

În cazul în care N^{M_i} este finit, valoarea momentană a acestuia poate fi controlată din două perspective:

i.) În timpul execuţiei lui M_i, executivul/dispatcher-ul sistemului de operare TR decrementează de fiecare dată pe N^{M_i} cu o unitate, până acesta ajunge la 0. În acest mod, sistemul contorizează execuţiile lui Mi pe parcursul operării. Când N^{M_i} ajunge la 0, executivul nu-l va mai lansa în execuţie pe M_i.

ii.) În timpul execuţiei unui alt ModX, M_j, acesta poate accesa contorul de execuţii al lui M_i, reactualizându-l la orice valoare posibilă, conform necesităţilor aplicaţiei. Prin acest mecanism, se poate relansa în execuţie un ModX, chiar dacă acesta are contorul setat în mod curent pe 0. Metoda presupune acordarea unei atenţii sporite fluxului de control al programului, care poate fi modificat într-un mod incontrolabil.

Definiţia 4-10.

Numim *ModX fantomă*, un ModX M_i, care la un moment dat are contorul de execuţii setat pe 0 ($N^{M_i} = 0$). Aşa cum s-a văzut mai sus, ModX-urile fantomă sunt planificate în cadrul sistemului TR, dar nu sunt lansate în execuţie.

❑

Figura 4-7 exemplifică operarea, pentru două perioade consecutive, a unui ModX oarecare M_i ce are specificaţi toţi parametrii săi temporali (*ModX generic*).

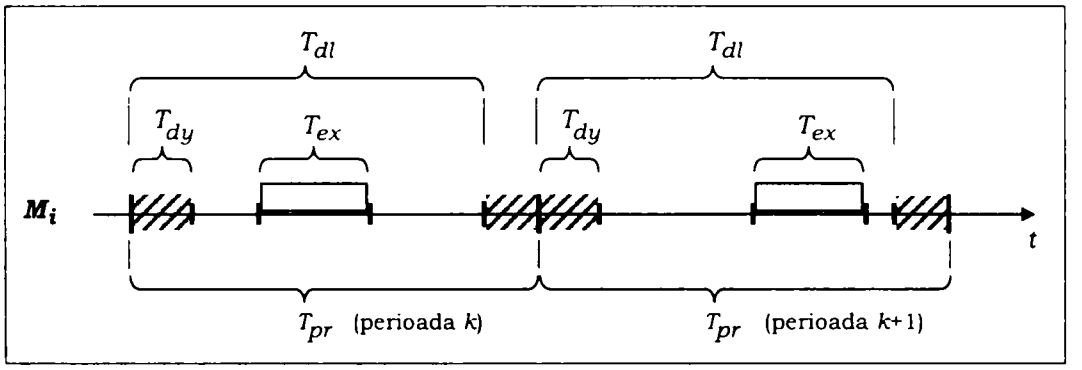

Figura 4-7. ModX-ul generic M_i şi parametrii sări temporali

Specificarea temporală minimală a unui ModX M_i, constă din cei doi parametri de bază: perioada $T_{pr}^{M_i}$ şi durata de execuţie $T_{ex}^{M_i}$. Pe baza acestora, algoritmii de planificare pot analiza fezabilitatea setului de ModX-uri al sistemului TR strict şi, în caz că există o variantă de planificare, o pot obţine. Setarea celorlalţi parametri temporali se va face implicit, în faza de validare a aplicaţiei, după cum urmează:

$$\begin{cases} T_{dl}^{M_i} = T_{pr}^{M_i} \\ T_{dy}^{M_i} = 0 \\ N^{M_i} = \infty \end{cases} \tag{4-23}$$

Relaţia (4-23) specifică pentru M_i, un interval limită egal cu perioada, un interval de întârziere nul şi un număr nedefinit de execuţii (operare continuă). Figura 4-8 exemplifică operarea pentru două perioade consecutive a unui ModX M_i ce are specificaţi doar cei doi parametri temporali de bază (*ModX simplu*).

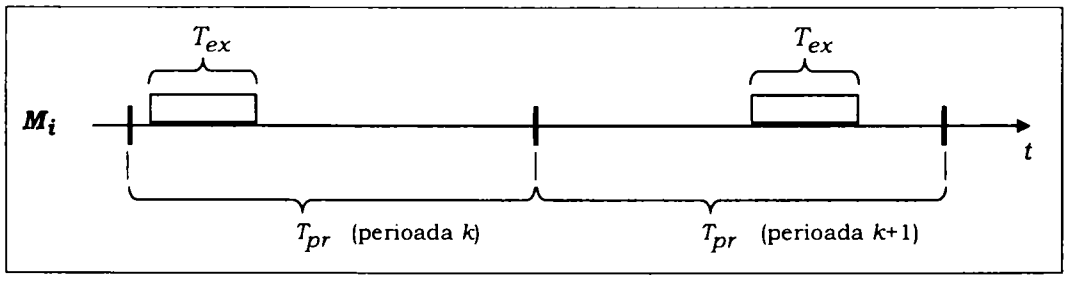

Figura 4-8. ModX-ul simplu M_i şi parametrii săi temporali

Verificarea şi analiza aplicaţiei TR din punct de vedere al comportamentului temporal se bazează pe următoarele relaţii stabilite între parametrii temporali ai ModX-urilor şi ai semnalelor utilizate în aplicaţie:

$$0 < T_{ex}^{M_i} \le T_{dl}^{M_i} \le T_{pr}^{M_i} \tag{4-24}$$

$$0 \le T_{dy}^{M_i} \le T_{dl}^{M_i} - T_{ex}^{M_i} < T_{dl}^{M_i} \le T_{pr}^{M_i} \tag{4-25}$$

$$T_{pr}^{M_i} \le \left\lfloor \frac{\min\left(T_{rs}^{S_j}, T_{pr}^{S_j}\right)+1}{2} \right\rfloor \tag{4-26}$$

pentru $\forall M_i \in \boldsymbol{M}$ (vezi (4-20)) şi $S_j \in \boldsymbol{S}$ (vezi (4-21)).

Formulele (4-24), (4-25) şi (4-26) sunt variante ale relaţiilor (4-14), (4-15), respectiv (4-16) şi (4-18), adaptate definirii ModX-urilor ca task-uri TR stricte, periodice şi cu parametrii temporali (mulţimea $\boldsymbol{T}$ din (4-21)) constanţi pentru toate ciclurile de execuţie.

5 Planificarea aplicațiilor TR stricte

5.1 Introducere

Una dintre problemele de interes major în dezvoltarea şi analiza sistemelor şi aplicațiilor TR o reprezintă planificarea task-urilor. În literatura curentă există studii teoretice şi simulări efectuate asupra unei multitudini de algoritmi de planificare pentru STR [Mercer 92, Panzieri 93a, Panzieri 93b, Ghosh 94, Ramamritham 94, George 96, Letia 00, Radulescu 02] (vezi Capitolul 2).

O metodă de clasificare a algoritmilor de planificare este în funcție de admiterea sau nu a întreruperilor în timpul execuției task-urilor sistemului. Rezultă cele două categorii principale de algoritmi de planificare: algoritmii *preemptivi*, care acceptă întreruperea unui task aflat în execuție de către alte task-uri, cu prioritate mai mare de exemplu, şi algoritmii *non-preemptivi*, care nu acceptă întreruperi în timpul execuției unui task. Algoritmii de planificare preemptivă au fost modelați şi tratați în numeroase lucrări, din care menționăm pe [Liu 73] ca fiind de referință, în timp ce algoritmii non-preemptivi nu s-au bucurat de aceeaşi atenție, în special din cauză că generează o mulțime de probleme pentru care s-a demonstrat că necesită resurse (timpi) de rezolvare de factură strict nepolinomială (*NP-hard*) [Cheng 88, Panzieri 93b]. Cu toate acestea, algoritmii de planificare non-preemptivă sunt importanți dintr-o varietate de motive [Jeffay 91]:

- Pentru multe probleme practice de planificare a task-urilor TR, cum ar fi de exemplu, planificarea task-urilor ce interacționează cu semnale de intrare/ieşire ale sistemului, proprietățile hardware şi software ale cuplului STR – dispozitiv periferic nu permit lucrul cu întreruperile sau acesta devine mult prea costisitor;
- Algoritmii de planificare non-preemptivi sunt mai uşor de implementat decât cei preemptivi şi, în cele mai multe cazuri, au ca efect o scădere drastică a resurselor suplimentare necesare planificării din timpul operării sistemului (*run-time scheduling*);
- Comportamentul şi resursele de timp şi de calcul al algoritmilor de planificare preemptivă sunt mai dificil de caracterizat şi de modelat, spre deosebire de cei non-preemptivi. Mai mult – în cele mai frecvente abordări, resursele sistem ocupate în timpul planificării (*run-time scheduling overhead*) sunt ignorate, rezultând că implementarea unui algoritm non-preemptiv este mai apropiată de modelul formal studiat decât cazul unui algoritm preemptiv;
- Aşa cum s-a văzut şi în Capitolul 4, task-urile non-preemptive, planificate pe un STR mono-procesor, garantează accesul exclusiv la resursele partajate ale sistemului, eliminând astfel necesitatea şi costurile implementării mecanismelor de sincronizare.

În completarea celor de mai sus, aşa cum remarcă [Gai 02], planificarea non-preemptivă este o metodă naturală şi directă, în special în cazul arhitecturilor de sistem şi a aplicaţiilor TR de achiziţie şi prelucrare numerică a datelor în ritm continuu, neîntrerupt (de exemplu, în aplicaţiile TR ce au la bază procesoare numerice de semnal, DSP).

Există însă şi o serie de dezavantaje ale algoritmilor de planificare non-preemptivă. Un prim neajuns a fost menţionat mai sus – generarea de probleme computaţionale de ordin strict nepolinomial. Alte dezavantaje includ:

- Necesitatea găsirii unei partajări optime a secvenţei de cod a programului în task-uri (problema *granularităţii* task-urilor), în aşa fel încât planificarea non-preemptivă a acestora să poată fi efectuată în condiţii optime de eficienţă. Dacă se lucrează cu o granularitate prea mică (task-uri de dimensiuni mari), planificarea non-preemptivă devine extrem de dificilă. Dacă, dimpotrivă, se lucrează cu o granularitate prea mare, se complică foarte mult mecanismele de transmitere a parametrilor şi de salvare/restaurare a contextului între task-urile aplicaţiei;
- Lipsa de flexibilitate a abordării non-preemptive. Odată cu eliminarea întreruperilor din sistem, se elimină şi capacitatea sistemului de a face faţă unor configuraţii noi, asincrone, ale mediului în care operează şi cu care interacţionează. Din aceste motive, vor trebui prevăzute şi tratate apriori, încă din faza de specificare şi proiectare, toate posibilităţile şi configuraţiile de operare ale sistemului;
- Creşterea ineficienţei sistemului, comparativ cu cazurile de planificare preemptivă. Mecanismul întreruperilor a fost conceput tocmai pentru a permite operarea cât mai eficientă a sistemelor digitale, mai ales a celor care interacţionează cu semnale (evenimente) asincrone;
- Dificultatatea găsirii unor condiţii de testare rapidă a planificabilităţii şi de găsire a planificării optime a unui set de task-uri. Pentru multe modele ale aplicaţiilor, testarea planificabilităţii setului aferent de task-uri şi găsirea unui algoritm optim de planificare reprezintă probleme computaţionale de ordin strict nepolinomial.

Avantajul major al algoritmilor de planificare non-preemptivă constă însă în faptul că permit operarea STR (mai ales a sistemelor stricte) în condiţii maxime de predictibilitate şi determinism, cu garantarea respectării termenelor de timp impuse task-urilor TR stricte, spre deosebire de algoritmii de planificare ce permit existenţa întreruperilor, a evenimentelor şi task-urilor asincrone, sporadice.

Exemplul 5-1.

Considerăm o staţie terestră de emisie-recepţie care comunică permanent cu un satelit nestaţionar aflat pe o orbită în jurul Pământului. Două dintre funcţiunile principale executate de sistemul digital de control şi procesare al staţiei sunt comunicaţia de date cu satelitul şi comanda dispozitivelor de orientare a antenei parabolice în aşa fel încât să fie permanent şi perfect aliniată cu satelitul.

Ambele funcţii se desfăşoară în regim permanent, dar, dacă partea de comunicaţie poate fi modelată în mod direct ca un task periodic (set de task-uri

periodice), funcţia de orientare a antenei poate fi modelată ca un task asincron: necesitatea reorientării antenei fiind semnalată de scăderea sub un anumit prag a intensităţii semnalului recepţionat de la satelit. Evident, sistemul de control şi procesare al staţiei este un sistem TR strict.

Dacă sistemul implementează un algoritm de planificare preemptiv, pot apărea situaţii în care, prin întreruperea execuţiei unuia din cele două task-uri de către celălalt, să rezulte pierderea orientării antenei sau a unor pachete de date vehiculate pe legătura de comunicaţie dintre sistemul terestru şi satelit.

Pe de altă parte, abordarea non-preemptivă a problemei presupune o analiză minuţioasă, prealabilă, a planificării şi execuţiei celor două task-uri: cel de orientare şi cel de comunicaţie. Task-ul ce tratează comunicaţia poate fi modelat ca un task periodic. Task-ul ce tratează orientarea antenei poate fi modelat, de asemenea, ca un task periodic, ţinând cont de intervalul de timp minim în care poate apărea necesitatea orientării antenei (parametru ce depinde de viteza satelitului, distanţa între satelit şi antenă, etc.). Analiza planificabilităţii non-preemptive a setului compus din cele două task-uri se bazează pe perioadele şi pe duratele lor de execuţie, verificându-se respectarea termenelor şi evitarea suprapunerii execuţiilor în timpul operării.

În cazul în care analiza de fezabilitate a planificării non-preemptive a task-urilor sistemului dă rezultate pozitive, există garanţia operării predictibile şi cu respectarea strictă a termenelor specificate.

❑

În capitolul de faţă vom aborda problema planificării non-preemptive a task-urilor TR stricte în *sisteme TR monoprocesor*, tratând diverse modele posibile, de la cele mai simple, până la modele mai complexe şi mai apropiate de realitate. Vom pleca de la următoarele caracteristici iniţiale, comune tuturor cazurilor pe care le vom analiza în continuare:

i.) Se va studia problema planificării non-preemptive monoprocesor a task-urilor TR stricte, periodice;
ii.) Modelul task-urilor TR stricte se bazează pe caracteristicile generale ale ModX-urilor (vezi Capitolul 4): task-uri periodice, cu durată de execuţie constantă de la un ciclu de execuţie la altul;
iii.) Înainte de execuţia efectivă a sistemului de task-uri în cadrul STR, are loc o analiză a fezabilităţii planificării acestora. Numim această operaţie *analiză offline*;
iv.) STR operează într-un mediu determinist şi toate specificaţiile temporale necesare sunt cunoscute;
v.) Modelul temporal utilizat este cel descris în Capitolul 4: un domeniu temporal discret, limitat la stânga, nelimitat la dreapta şi cu metrică temporală;
vi.) Toţi algoritmii de planificare studiaţi vor implementa o *planificare fără inserţie de timp liber*, adică nu vor introduce în planificare intervale de timp liber în situaţiile în care există task-uri invocate de către sistem.

5.2 Planificarea non-preemptivă a ModX-urilor independente

Majoritatea abordărilor din literatură, legate de planificarea non-preemptivă, se concentrează pe modelul cel mai simplu şi mai apropiat de ideal, în ceea ce priveşte setul de task-uri analizat: modelul task-urilor TR stricte, independente [Kim 80, Jeffay 91, Howell 95, George 96, Kang 98].

5.2.1 Modelul setului de task-uri independente

Modelul setului de task-uri cu ajutorul căruia studiem planificarea non-preemptivă a task-urilor independente este definit după cum urmează.

Definiţia 5-1.
Definim setul $\boldsymbol{M}$ de ModX-uri simple, independente, ca fiind mulţimea

$$\boldsymbol{M} = \{M_1, M_2, \ldots, M_n\}$$

unde, fiecare element M_i al lui $\boldsymbol{M}$ ($1 \le i \le n$) este un ModX, cu următoarele caracteristici:

- M_i este un ModX simplu (un task TR strict, periodic, planificat şi executat în context non-preemptiv).
- Din punct de vedere al comportamentului în timp, M_i este caracterizat prin următorii parametri temporali:

$$\boldsymbol{T} = \left\{ T_{pr}^{M_i}, T_{ex}^{M_i}, T_{dl}^{M_i}, T_{dy}^{M_i}, N^{M_i} \right\} \tag{5-1}$$

- $T_{pr}^{M_i}$ este perioada lui M_i, $T_{ex}^{M_i}$ este durata de execuţie, $T_{dl}^{M_i}$ este intervalul limită. $T_{dy}^{M_i}$ este intervalul de întârziere şi N^{M_i} este numărul total de execuţii.
- Între parametrii lui M_i se stabilesc următoarele relaţii:

$$0 < T_{ex}^{M_i} \le T_{pr}^{M_i} \tag{5-2}$$

$$T_{dl}^{M_i} = T_{pr}^{M_i} \tag{5-3}$$

$$T_{dy}^{M_i} = 0 \tag{5-4}$$

$$N^{M_i} = \infty \tag{5-5}$$

- Momentul primei invocări a fiecărui ModX M_i este:

$$t_{rq,1}^{M_i} = 0 \tag{5-6}$$

- Execuţia lui M_i nu este condiţionată de execuţiile altor ModX-uri, adică nu există dependeţe de control sau de date între oricare două ModX-uri din ***M***.

❑

Setul ***T*** al parametrilor temporali din (5-1) specifică un ModX generic, pentru care, în mod obligatoriu, durata de execuţie este nenulă şi mai mică sau egală cu perioada. Relaţiile (5-2) până la (5-4) identifică un ModX simplu (vezi Capitolul 4): intervalul limită (*deadline*) este egal cu perioada, intervalul de întârziere este nul, respectiv, numărul de cicluri de execuţie este nelimitat.

Definiţia 5-2.

Spunem că setul de ModX-uri ***M*** este *fezabil din punct de vedere al planificării*, dacă există un algoritm capabil să planifice toate ModX-urile din ***M*** în aşa fel încât fiecare să-şi respecte specificaţiile de comportare temporală pe toată durata operării sistemului.

❑

Există două clase de algoritmi de planificare ce pot fi aplicaţi pentru seturi de ModX-uri independente:

- *Algoritmi de planificare statică*: asignează câte o prioritate fixă de planificare pentru fiecare ModX după un anumit criteriu. La o instanţă de timp oarecare, t_x (pentru care procesorul este liber şi se poate planifica un ModX), va fi selectat ModX-ul cu prioritatea cea mai mare şi care nu a mai fost executat în perioada corespunzătoare lui t_x. De exemplu, un criteriu pentru stabilirea priorităţilor în cadrul unui set de ModX-uri este asignarea unei priorităţi invers proporţionale cu lungimea perioadei fiecăruia. Cea mai cunoscută tehnică de planificare statică este *RMS* (*Rate Monotonic Scheduling*) [Liu 73, Kim 80, George 96].
- *Algoritmi de planificare dinamică*: selectarea ModX-ului care va fi planificat la un moment dat t_x, se face în mod dinamic după un criteriu ce include, printre altele, valoarea lui t_x şi cerinţa ca ModX-ul să nu fi fost executat în perioada corespunzătoare lui t_x. De exemplu, un criteriu utilizat în planificarea dinamică este selectarea ModX-ului a cărui perioadă se termină cel mai aproape de momentul t_x (sau, cu alte cuvinte, ModX-ul al cărui deadline este cel mai apropiat de t_x). Algoritmul de planificare ce utilizează acest criteriu este *EDF* (*Earliest Deadline First*) [Liu 73, Kim 80, Jeffay 91, Howell 95, George 96, Kang 98].

În continuare vom analiza doi dintre algoritmii de planificare dinamică, consideraţi a fi cei mai eficienţi pentru tratarea execuţiilor non-preemptive [Kim 80, Jeffay 91, Howell 95, George 96]. Este vorba de algoritmul *MLFNP* (*Minimum Laxity First Non-Preemptive*, sau *LLFNP – Least Laxity First Non-Preemptive*) şi *EDFNP* (*Earliest Deadline First Non-Preemptive*).

Lema 5-1.

Considerăm un set ***M*** de ModX-uri cu caracteristicile date de Definiţia 5-1 şi notăm cu T_{LCM}, intervalul de timp egal cu cel mai mic multiplu comun al perioadelor ModX-urilor din ***M***:

$$T_{LCM} = \min\left\{ C \,\middle|\, T_{pr}^{M_i} \,/ C,\ \forall M_i \in \boldsymbol{M} \right\} \tag{5-7}$$

unde, cu x / y s-a notat faptul că x divide pe y.

Dacă un algoritm este capabil a planifica ModX-urile din ***M*** în intervalul T_{LCM}, atunci setul ***M*** este fezabil din punctul de vedere al planificării cu acest algoritm.

Demonstraţie

Lema susţine că dacă un algoritm este capabil să planifice cu succes ModX-urile din ***M*** pe un interval de timp egal cu cel mai mic multiplu comun al perioadelor ModX-urilor, atunci planificarea setului ***M*** va reuşi pe toată durata operării sistemului.

Setul ***M*** conţine ModX-uri simple, cu caracteristicile date de Definiţia 5-1. Astfel, pentru toate ModX-urile, prima invocare are loc la momentul 0 (5-6), iar comportamentul lor în timp este complet determinat de către perioada şi durata fiecăruia de execuţie: $T_{pr}^{M_i}$, respectiv $T_{ex}^{M_i}$, pentru $\forall M_i \in \boldsymbol{M}$.

Din (5-6) rezultă că perioadele tuturor ModX-urilor din ***M*** sunt aliniate la momentul 0, şi, mai mult, ele sunt de asemenea aliniate la fiecare moment de timp care este un multiplu comun al acestor perioade. Cu alte cuvinte, putem spune că, din punct de vedere al perioadelor, setul ***M*** are o configuraţie ciclică, cu perioada de bază T_{LCM}.

Pe de altă parte, algoritmii de planificare vizează ca nici un ModX din ***M*** să nu-şi depăşească termenele specificate, deci ca fiecare ModX să fie executat o dată (şi numai odată), pentru fiecare perioadă a sa, de-a lungul operării sistemului.

Rezultă din cele de mai sus că se poate stabili o comportare ciclică şi pentru algoritmii de planificare, pe baza intervalului T_{LCM}.

❑

Lema 5-1 permite efectuarea analizei offline a fezabilităţii planificării unui set de ModX-uri independente doar pentru un interval de timp de lungime T_{LCM}. Astfel, dacă un anume algoritm de planificare dă rezultate pozitive la planificarea unui set de ModX-uri pe intervalul $[0, T_{LCM})$, atunci setul respectiv este fezabil.

Indiferent de criteriul de selecţie al ModX-ului care va fi planificat la un moment de timp dat, t, algoritmii de planificare dinamică, non-preemptivă, a ModX-urilor independente operează după organigrama generală reprezentată în Figura 5-1.

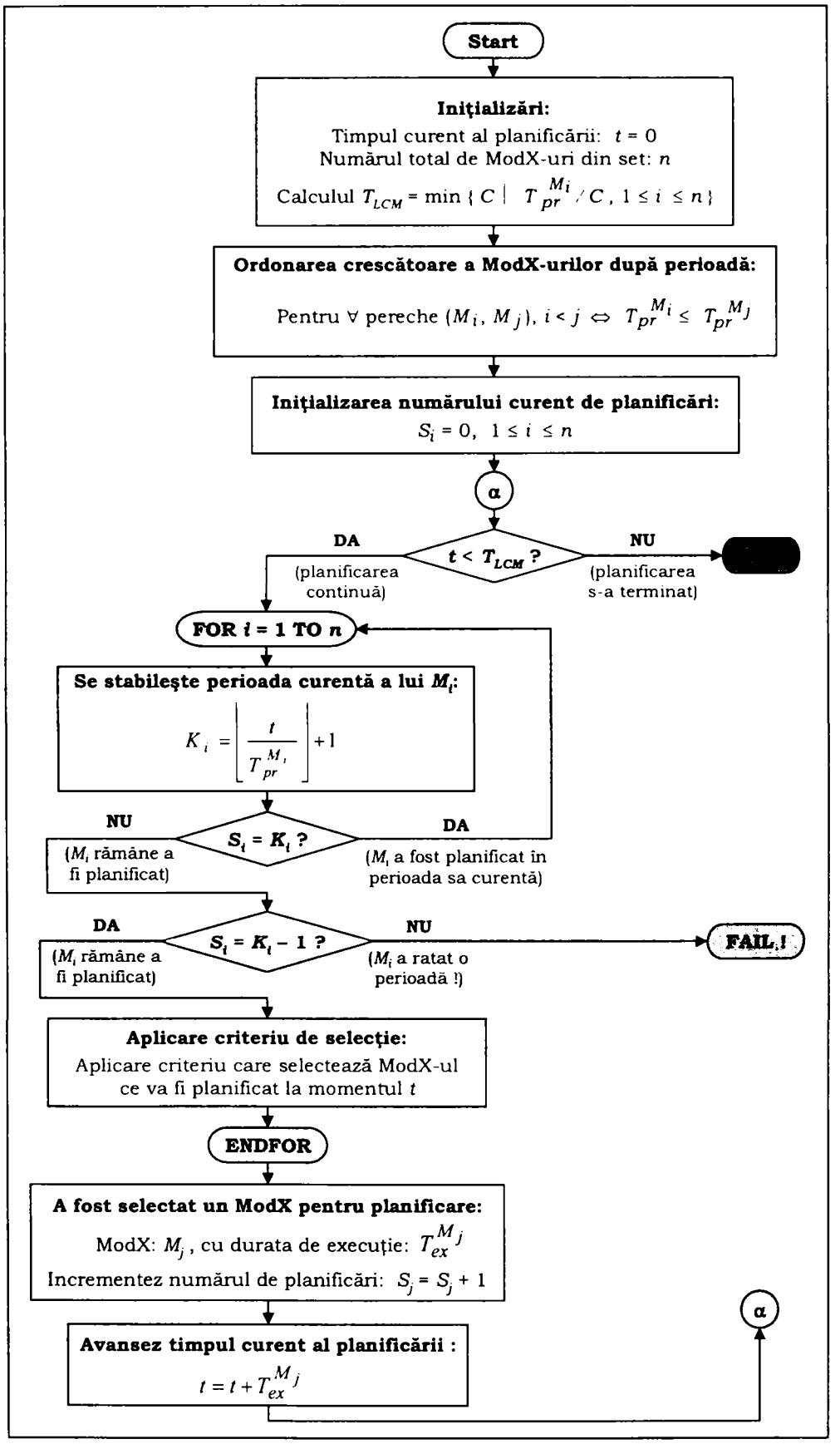

Figura 5-1. Organigrama generală a algoritmilor de planificare non-preemptivă a unui set de ModX-uri independente

5.2.2 Algoritmul de planificare MLFNP

Pentru selecţia ModX-ului ce va fi planificat la momentul *t*, algoritmul de planificare *MLFNP* (*Minimum Laxity First Non-Preemptive*) utilizează un criteriu bazat pe intervalul disponibil planificării (*Laxity interval*, vezi Tabelul 4-3).

În cazul ModX-urilor simple, intervalul disponibil planificării, notat cu $T_{lx}^{M_i}$ (pentru ModX-ul M_i) este (vezi Figura 5-2):

$$T_{lx}^{M_i} = T_{pr}^{M_i} - T_{ex}^{M_i}, \forall M_i \in \boldsymbol{M} \tag{5-8}$$

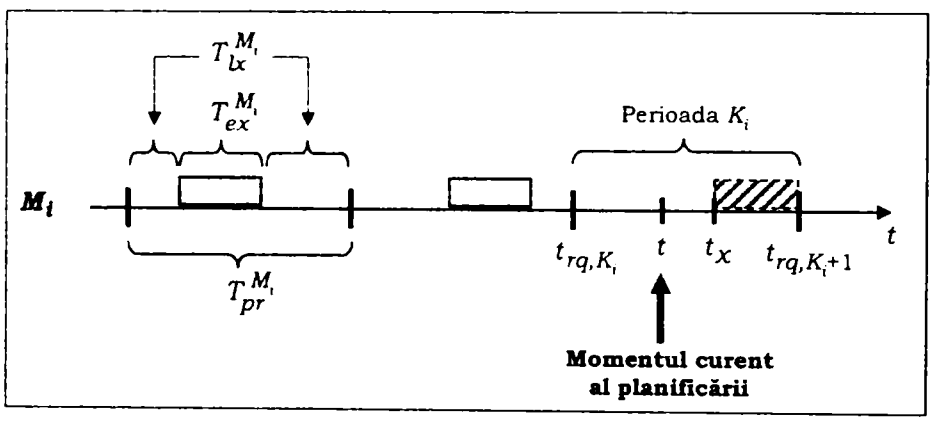

Figura 5-2. Intervalul disponibil planificării în cazul ModX-urilor simple

Din (5-8) rezultă că $T_{lx}^{M_i}$ este un parametru constant, ce nu depinde de numărul ciclului curent de execuţie (planificare), K_i, a lui M_i, ci doar de perioada şi de durata sa de execuţie.

Criteriul de selecţie al ModX-ului ce va fi planificat la un moment *t* se bazează pe ideea de interval de timp pe care îl mai are la dispoziţie algoritmul, din momentul *t*, până la depăşirea termenelor fiecărui ModX. Cu cât acest interval este mai mare pentru un ModX M_i, cu atât prioritatea (sau "urgenţa") planificării lui M_i este mai mică în comparaţie cu a altor ModX-uri, şi reciproc.

Pentru exemplul prezentat în Figura 5-2, la momentul curent al planificării, *t*, ModX-ul M_i se găseşte în al K_i-lea ciclu de execuţie, delimitat de momentele t_{rq,K_i} şi t_{rq,K_i+1}, astfel încât:

$$\left[t_{rq,K_i}, t_{rq,K_i+1}\right] = T_{pr}^{M_i} = t_{rq,K_i+1} - t_{rq,K_i}.$$

Determinarea lui K_i se face pentru fiecare ModX M_i din setul $\boldsymbol{M}$, în cadrul buclei de selecţie a ModX-ului ce va fi planificat, şi pentru fiecare moment *t* al planificării (vezi Figura 5-1):

$$K_i = \left\lfloor \frac{t}{T_{pr}^{M_i}} \right\rfloor + 1 \tag{5-9}$$

Pentru planificarea corectă a lui M_i în ciclul K_i, algoritmul mai are la dispoziţie intervalul de timp $[t, t_x]$ (vezi Figura 5-2). Dacă notăm lungimea acestui interval cu $L_i(t)$, avem că:

$$L_i(t) = t_x - t = K_i \cdot T_{pr}^{M_i} - T_{ex}^{M_i} - t \tag{5-10}$$

şi din (5-9), rezultă:

$$L_i(t) = T_{pr}^{M_i}\left(\left\lfloor \frac{t}{T_{pr}^{M_i}} \right\rfloor + 1\right) - T_{ex}^{M_i} - t \tag{5-11}$$

Evident, $L_i(t)$ e o funcţie de timp (şi anume, de momentul curent al planificării) şi depinde de parametrii temporali ai lui M_i. Rezultă că la un moment dat, algoritmul *MLFNP* va calcula maxim n astfel de funcţii, câte una pentru fiecare ModX din $\boldsymbol{M}$ (unde $n = |\boldsymbol{M}|$). În finalul buclei de selecţie, algoritmul va planifica ModX-ul al cărui interval disponibil pentru planificare este cel mai mic:

$$M_i \text{ - planificat la momentul } t \quad \Leftrightarrow L_i(t) = \min\left\{L_j(t) \mid 1 \le j \le n\right\} \tag{5-12}$$

Formula (5-12) exprimă formal criteriul utilizat de algoritmul *MLFNP* pentru selecţia ModX-ului ce va fi planificat la momentul curent t. După ce ModX-ul M_i a fost selectat şi planificat, algoritmul incrementează variabila corespunzătoare numărului de planificări a lui M_i (vezi organigrama din Figura 5-1) şi avansează momentul următoarei planificări cu durata de execuţie a lui M_i:

$$\begin{cases} S_i = S_i + 1 \\ t = t + T_{ex}^{M_i} \end{cases},$$

după care reia procedura, de la noul moment t. Algoritmul se termină în momentul în care planificarea s-a desfăşurat cu succes pe intervalul $[0, T_{LCM})$.

În continuare vom exemplifica operarea algoritmului *MLFNP* pentru câteva cazuri concrete de seturi de ModX-uri.

Exemplul 5-2.

Considerăm un set $\boldsymbol{M} = \{M_1, M_2, M_3, M_4\}$ de ModX-uri simple, independente, caracterizate prin următorii parametri temporali:

$$M_1 : \begin{cases} T_{pr}^{M_1} = 8 \\ T_{ex}^{M_1} = 2 \end{cases}; \quad M_2 : \begin{cases} T_{pr}^{M_2} = 9 \\ T_{ex}^{M_2} = 4 \end{cases}; \quad M_3 : \begin{cases} T_{pr}^{M_3} = 18 \\ T_{ex}^{M_3} = 3 \end{cases}; \quad M_4 : \begin{cases} T_{pr}^{M_4} = 24 \\ T_{ex}^{M_4} = 3 \end{cases}$$

Se doreşte analiza offline a planificabilităţii setului $\boldsymbol{M}$, utilizând algoritmul *MLFNP*. Figura 5-3 ilustrează rezultatul planificării, pentru intervalul de timp $[0, T_{LCM})$, unde $T_{LCM} = 72$.

Planificarea începe de la momentul $t = 0$, după iniţializarea parametrului S_i (numărul total de planificări pentru ModX-ul M_i, până la momentul curent):

$$S_1 = S_2 = S_3 = S_4 = 0$$

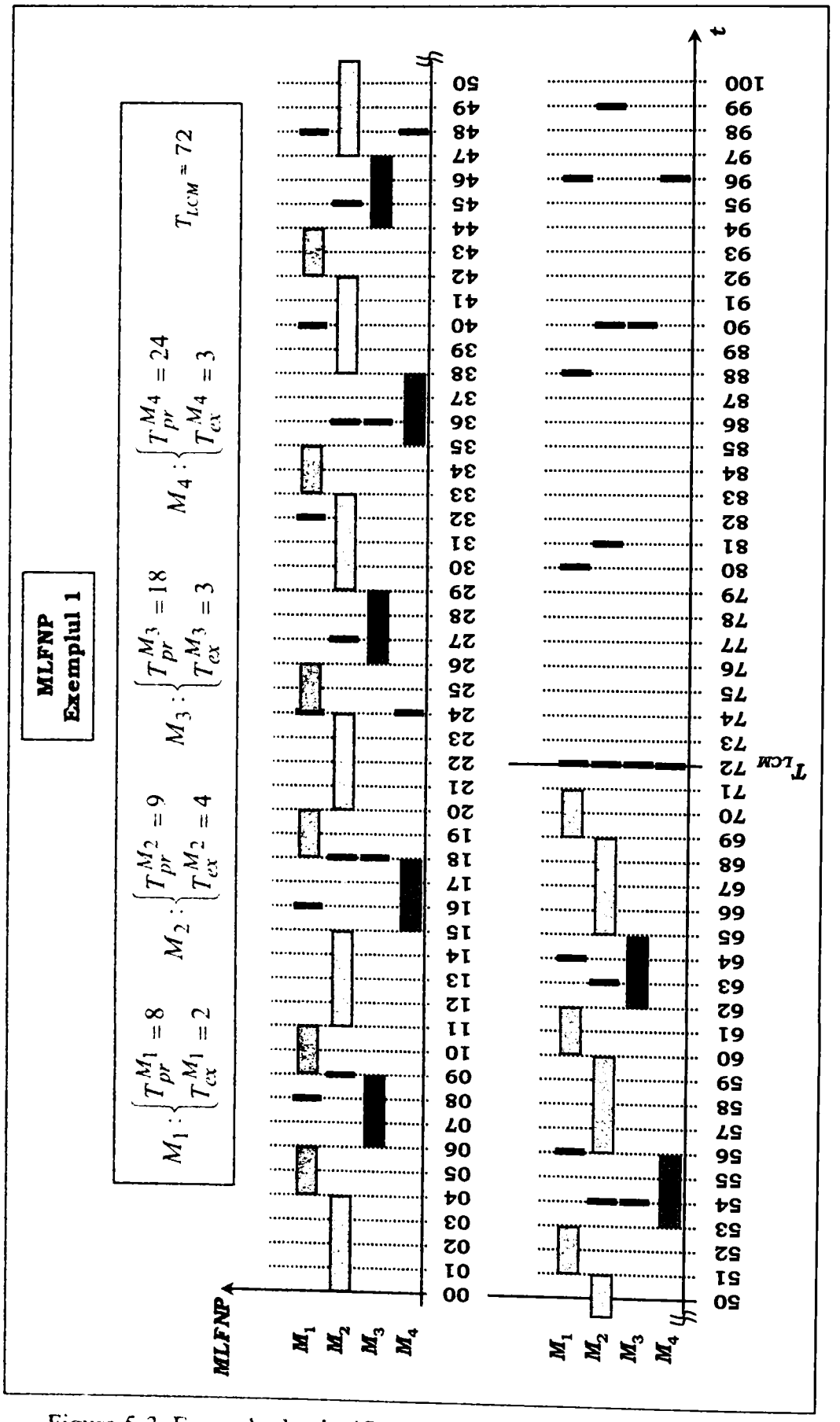

Figura 5-3. Exemplu de planificare *MLFNP* a unui set de ModX-uri

Primul pas îl reprezintă stabilirea pentru fiecare ModX a numărului ciclului de execuţie K_i, corespunzător momentului curent t, aplicând (5-9):

$$K_1 = K_2 = K_3 = K_4 = 1$$

În continuare, se calculează intervalul de timp disponibil planificării fiecărui ModX, $L_i(t)$, conform relaţiei (5-11):

$$\begin{cases} L_1(0) = 8 \cdot \left(\left\lfloor \frac{0}{8} \right\rfloor + 1 \right) - 2 - 0 = 6 \\ L_2(0) = 9 \cdot \left(\left\lfloor \frac{0}{9} \right\rfloor + 1 \right) - 4 - 0 = 5 \\ L_3(0) = 18 \cdot \left(\left\lfloor \frac{0}{18} \right\rfloor + 1 \right) - 3 - 0 = 15 \\ L_4(0) = 24 \cdot \left(\left\lfloor \frac{0}{24} \right\rfloor + 1 \right) - 3 - 0 = 21 \end{cases}$$

de unde rezultă că $L_i(0)$ este minim pentru $i = 2$, deci M_2 va fi planificat la momentul $t = 0$. Ca urmare, se incrementează S_2 ($S_2 = 1$) şi se avansează timpul planificării:

$$t = t + T_{ex}^{M_2} = 4.$$

Procedura de planificare se reia de la $t = 4$, până când se ajunge la momentul $t = T_{LCM} = 72$. Planificarea setului $\boldsymbol{M}$ din acest exemplu dă rezultate pozitive, aşa cum se poate vedea şi în Figura 5-3.

❑

Exemplul 5-3.

Considerăm un set $\boldsymbol{M} = \{M_1, M_2, M_3\}$ de ModX-uri simple, independente, caracterizate prin următorii parametri temporali:

$$M_1 : \begin{cases} T_{pr}^{M_1} = 8 \\ T_{ex}^{M_1} = 3 \end{cases}; \quad M_2 : \begin{cases} T_{pr}^{M_2} = 10 \\ T_{ex}^{M_2} = 6 \end{cases}; \quad M_3 : \begin{cases} T_{pr}^{M_3} = 40 \\ T_{ex}^{M_3} = 1 \end{cases}$$

Se doreşte analiza offline a planificabilităţii setului $\boldsymbol{M}$, utilizând algoritmul *MLFNP*. Figura 5-4 ilustrează rezultatul planificării, pentru intervalul de timp $[0, T_{LCM})$, unde $T_{LCM} = 40$.

Algoritmul începe planificarea la $t = 0$, cu:

$$\begin{cases} L_1(0) = 8 - 3 = 5 \\ L_2(0) = 10 - 6 = 4 \\ L_3(0) = 40 - 1 = 39 \end{cases}$$

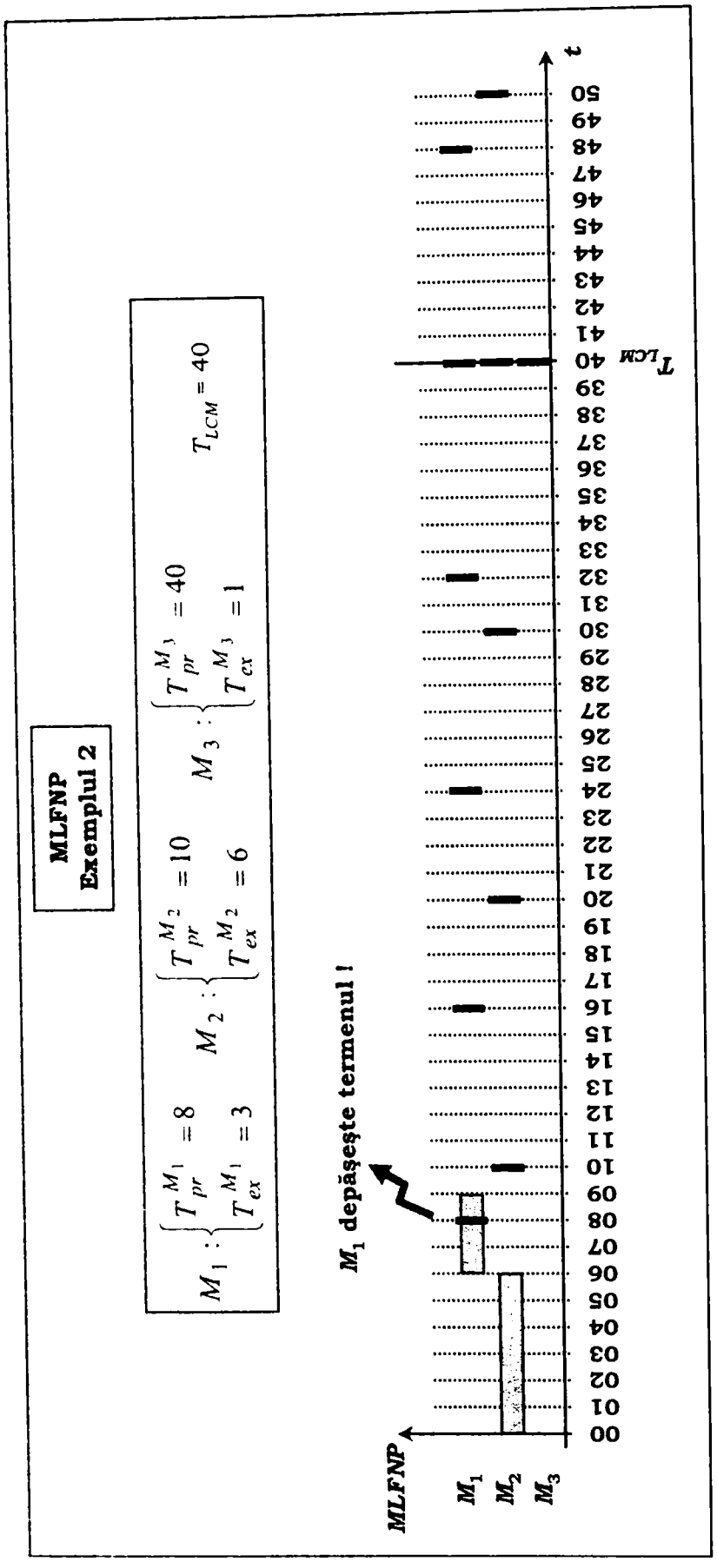

Figura 5-4. Exemplu de planificare MLFNP a unui set de ModX-uri

Aşa cum rezultă din Figura 5-4, setul ***M*** de ModX-uri nu poate fi planificat utilizând algoritmul *MLFNP* de planificare.

❑

Setul de ModX-uri din Exemplul 5-3 nu poate fi planificat utilizând algoritmul MLFNP (ceea ce nu înseamnă că nu poate fi planificat cu ajutorul unui alt algoritm – aşa cum se va vedea în secţiunea următoare). Vom spune că setul de ModX-uri din Exemplul 5-3 *nu este fezabil din punct de vedere al planificării* cu algoritmul MLFNP.

5.2.3 Algoritmul de planificare EDFNP

Planificarea *EDFNP* (*Earliest Deadline First Non-Preemptive*) utilizează un criteriu mai simplu pentru selectarea task-ului ce urmează a fi planificat la momentul t: se va selecta de fiecare dată task-ul cu termenul limită cel mai apropiat de t.

[Jeffay 91] demonstrează faptul că *EDFNP* este un algoritm optim de planificare, în sensul că dacă un set de ModX-uri este fezabil, atunci el poate fi planificat cu *EDFNP*. Exemplele următoare ilustrează acest lucru şi, în plus, arată că *EDFNP* este mai eficient ca *MLFNP* din cel puţin două motive:

- există seturi de ModX-uri care nu pot fi planificate cu *MLFNP*, dar *EDFNP* reuşeşte să le planifice,
- algoritmul *EDFNP* este mai simplu şi necesită resurse (de calcul) mai mici ca *MLFNP*.

Se vor utiliza aceleaşi seturi de ModX-uri ca în Exemplul 5-2 şi 5-3, pentru a ilustra inclusiv diferenţele de abordare a planificării (prin criteriile de selecţie a ModX-ului care urmează a fi planificat la momentul curent).

Exemplul 5-4.

Considerăm un acelaşi set $\boldsymbol{M} = \{M_1, M_2, M_3, M_4\}$ de ModX-uri simple, independente, ca în Exemplul 5-2:

$$M_1 : \begin{cases} T_{pr}^{M_1} = 8 \\ T_{ex}^{M_1} = 2 \end{cases}; \quad M_2 : \begin{cases} T_{pr}^{M_2} = 9 \\ T_{ex}^{M_2} = 4 \end{cases}; \quad M_3 : \begin{cases} T_{pr}^{M_3} = 18 \\ T_{ex}^{M_3} = 3 \end{cases}; \quad M_4 : \begin{cases} T_{pr}^{M_4} = 24 \\ T_{ex}^{M_4} = 3 \end{cases}$$

Figura 5-5 ilustrează analiza offline a planificabilităţii setului ***M***, utilizând algoritmul *EDFNP*, pentru intervalul de timp $[0, T_{LCM})$, unde $T_{LCM} = 72$.

La momentul $t = 0$, M_1 e ModX-ul cu termenul limită cel mai apropiat de t (intervalul minim de la t la momentul de invocare al perioadei următoare): $t_{rq,2}^{M_1} - t = 8$; $t_{rq,2}^{M_2} - t = 9$; $t_{rq,2}^{M_3} - t = 18$; $t_{rq,2}^{M_4} - t = 24$. După planificarea lui M_1 la momentul $t = 0$, se avansează timpul de planificare la $t = t + T_{ex}^{M_1} = 2$ şi se reia procedura de selecţie a ModX-urilor care nu au fost încă planificate.

❑

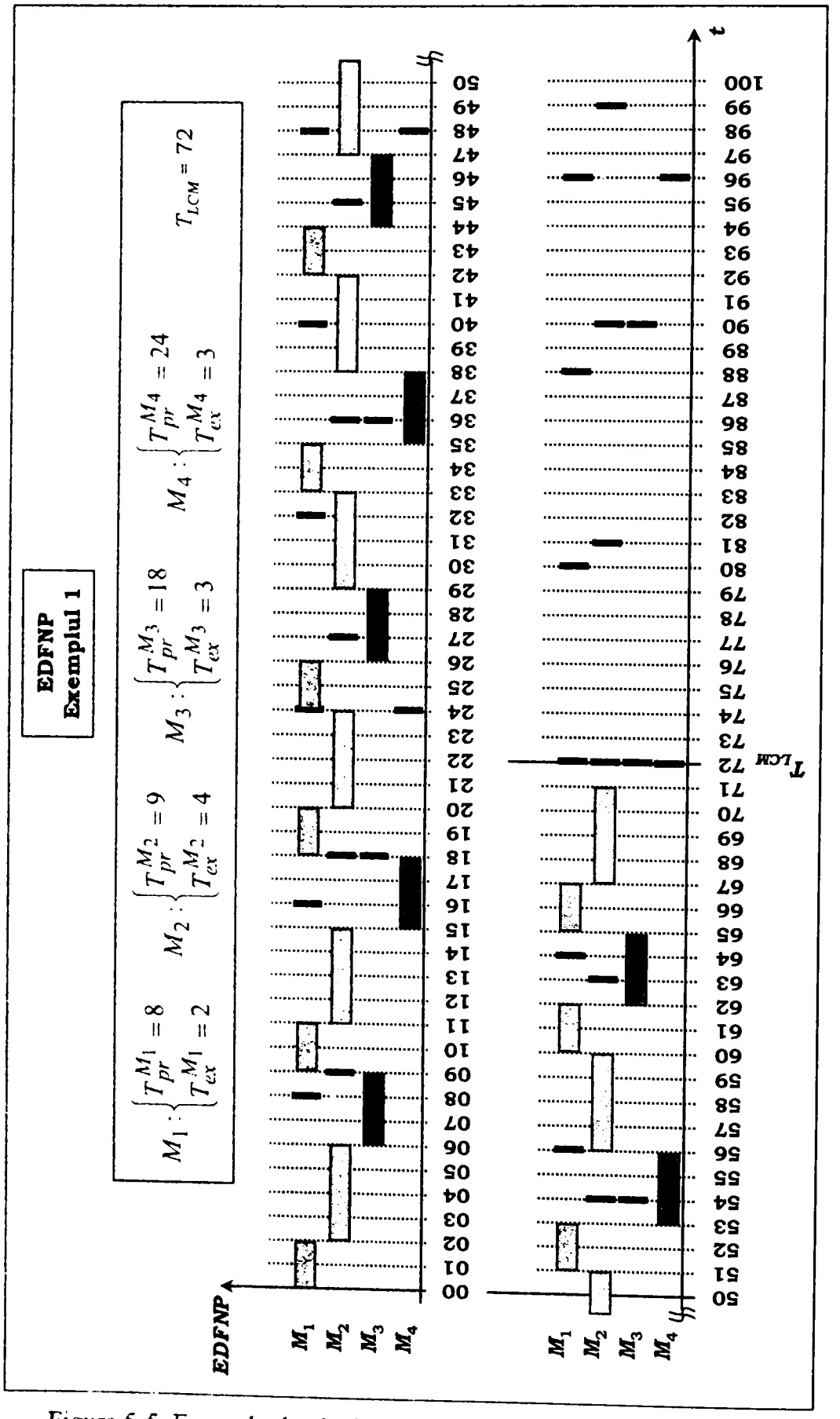

Figura 5-5. Exemplu de planificare *EDFNP* a unui set de ModX-uri

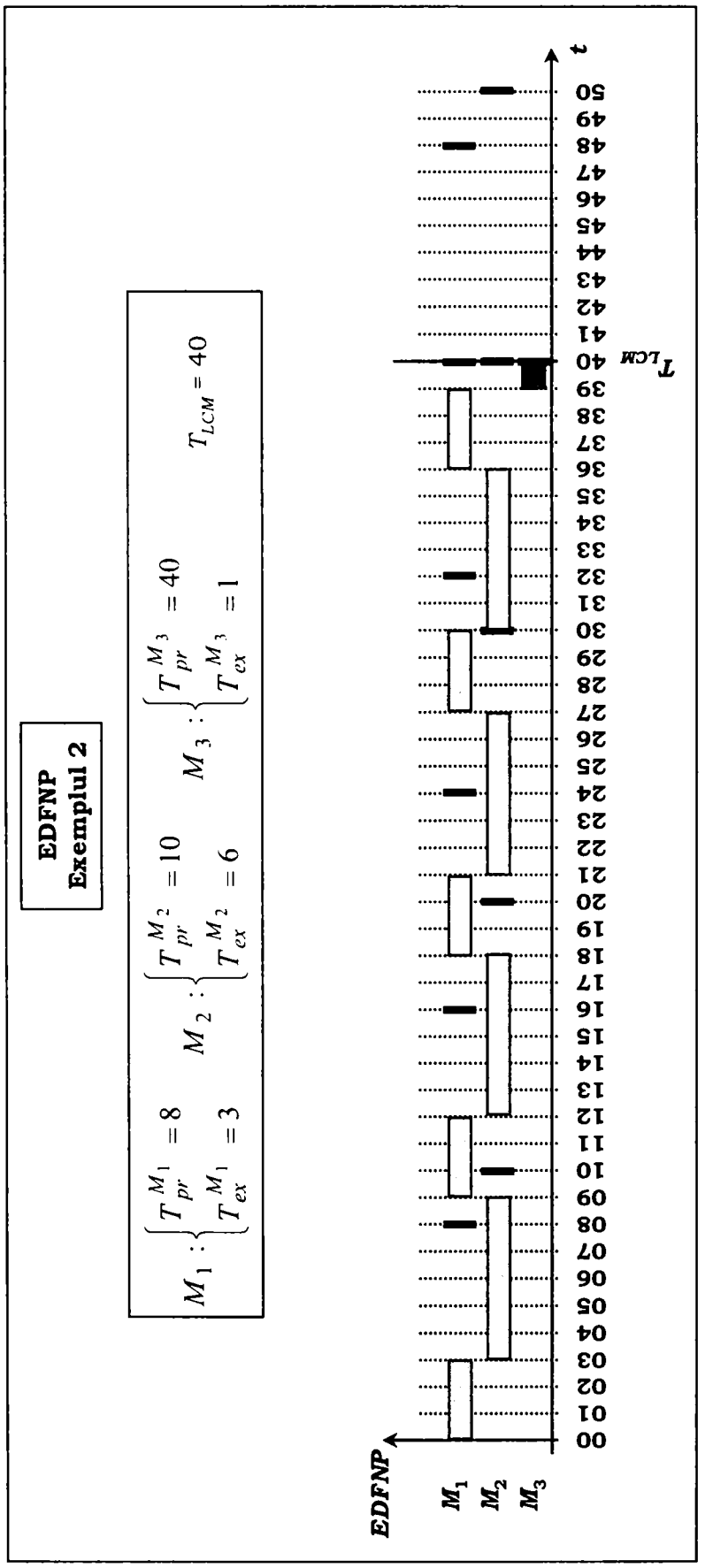

Figura 5-6. Exemplu de planificare *EDFNP* a unui set de ModX-uri

Exemplul 5-5.

Considerăm un acelaşi set $\boldsymbol{M} = \{M_1, M_2, M_3\}$ de ModX-uri simple, independente, ca în Exemplul 5-3:

$$M_1 : \begin{cases} T_{pr}^{M_1} = 8 \\ T_{ex}^{M_1} = 3 \end{cases}; \quad M_2 : \begin{cases} T_{pr}^{M_2} = 10 \\ T_{ex}^{M_2} = 6 \end{cases}; \quad M_3 : \begin{cases} T_{pr}^{M_3} = 40 \\ T_{ex}^{M_3} = 1 \end{cases}$$

Aşa cum rezultă şi din Figura 5-6, analiza offline a planificabilităţii setului $\boldsymbol{M}$ utilizând algoritmul *EDFNP* pentru intervalul $[0, T_{LCM})$, unde $T_{LCM} = 40$, dă rezultate pozitive, spre deosebire de cele obţinute cu *MLFNP*.

❑

5.2.4 Condiţiile de planificabilitate

În cadrul analizei offline a unei aplicaţii TR, determinarea fezabilităţii setului de task-uri din punct de vedere al planificării cu un anumit algoritm se poate realiza, fie aplicând direct algoritmul, fie evaluând mai întâi dacă setul de task-uri îndeplineşte o serie de condiţii – *condiţiile de planificabilitate*. Evaluarea condiţiilor de planificabilitate înaintea aplicării algoritmului propriu-zis prezintă avantajul unei decizii rapide, obţinute cu eforturi relativ mici (resurse de calcul, timp, etc.).

Condiţiile de planificabilitate reprezintă seturi de relaţii definite între parametrii temporali ai task-urilor din setul analizat, şi pot fi de două tipuri:

- *Condiţii de necesitate*: dacă setul de task-uri nu le îndeplineşte, atunci în mod sigur acesta nu este fezabil pentru planificare;
- *Condiţii de suficienţă*: dacă setul de task-uri le îndeplineşte, atunci în mod sigur acesta este fezabil şi va exista un algoritm care îl poate planifica cu succes.

Pentru cazul seturilor de task-uri TR stricte, ce vor fi planificate în context non-preemptiv, condiţiile de suficienţă sunt foarte dificil de determinat – dacă ele există [Kim 80, Jeffay 91]. Pot fi stabilite însă o serie de condiţii de necesitate, care să fie utile în identificarea rapidă a unui set de task-uri ce nu este fezabil. O primă condiţie de necesitate se referă la încărcarea procesorului [Liu 73, Kim 80, Jeffay 91, Howell 95, George 96, Kang 98], şi o a doua a fost enunţată şi demonstrată în [Jeffay 91]. Vom prezenta în continuare cele două condiţii de necesitate a planificabilităţii pentru cazul general al seturilor de task-uri ce vor fi planificate în context non-preemptiv şi le vom analiza sub aspectul valabilităţii şi aplicării lor în cazul modelului propus în lucrare.

Modelul de task-uri cu ajutorul căruia [Jeffay 91] studiază planificarea non-preemptivă se aseamănă cu modelul ModX-urilor independente propus în secţiunea de faţă: amândouă iau în considerare task-uri periodice, caracterizate din punct de vedere temporal prin doi parametri – perioada şi durata de execuţie. Diferenţa constă din faptul că în [Jeffay 91] se tratează seturile de task-uri la modul general, făcându-se distincţie dintre un "set de task-uri" şi un "set concret de task-uri". Dacă

$$\boldsymbol{M} = \{M_1, M_2, \ldots, M_n\}$$

este un set de task-uri, atunci un set concret de task-uri este mulţimea de perechi:

$$\boldsymbol{W} = \{(M_1, R_1), (M_2, R_2), ..., (M_n, R_n)\}$$

unde $R_i = t_{rq,1}^{M_i}$ reprezintă momentul primei invocări a task-ului M_i (*release time*). Un set de task-uri $\boldsymbol{M}$ poate genera o infinitate de seturi concrete de task-uri $\boldsymbol{W}$. Cu aceste premise, condiţiile de necesitate pentru planificarea non-preemptivă sunt enunţate în [Jeffay 91] după cum urmează:

"Fie $\boldsymbol{M} = \{M_1, M_2, ..., M_n\}$ un set de task-uri periodice sortate în ordine crescătoare a perioadelor (adică, pentru orice pereche de task-uri (M_i, M_j), dacă $i < j$, atunci $T_{pr}^{M_i} \le T_{pr}^{M_j}$).

Dacă $\boldsymbol{M}$ este planificabil, atunci:

CN1)
$$\sum_{i=1}^{n} \frac{T_{ex}^{M_i}}{T_{pr}^{M_i}} \le 1 \tag{5-13}$$

CN2) $\forall\, i,\ 1 < i \le n\ ;\ \forall\, L\ ,\ T_{pr}^{M_1} < L < T_{pr}^{M_i}$:

$$L \ge T_{ex}^{M_i} + \sum_{j=1}^{i-1} \left\lceil \frac{L-1}{T_{pr}^{M_j}} \right\rceil \cdot T_{ex}^{M_j} \tag{5-14}$$

unde fiecare task $M_i \equiv \left(T_{ex}^{M_i}, T_{pr}^{M_i}\right)$ este caracterizat prin parametrii săi temporali – durata de execuţie şi perioada."

CN1 este o condiţie de bază a oricărei planificări de task-uri pe sisteme cu un singur procesor şi exprimă cerinţa ca suma fracţiunilor de timp procesor consumate de fiecare task din setul $\boldsymbol{M}$ să nu depăşească unitatea, sau, cu alte cuvinte, *utilizarea procesor* (*processor utilization*) să fie mai mică sau egală cu 1. **CN1** poate fi aplicată şi modelului de set de task-uri bazat pe ModX-uri, propus în secţiunea de faţă. Cu ajutorul acestei condiţii, se pot elimina rapid acele seturi de ModX-uri pentru care utilizarea procesor e supraunitară, evitându-se astfel aplicarea algoritmului de planificare şi câştigându-se timp în faza de analiză offline.

Seturile de ModX-uri din Exemplul 5-2 şi 5-4, respectiv din Exemplul 5-3 şi 5-5 îndeplinesc **CN1**, având utilizarea procesor egală cu 0.986, respectiv 1.

Condiţia **CN2** reflectă restricţia impusă de planificarea non-preemptivă şi fără inserarea de timp liber pentru seturile de task-uri. Astfel, dacă un set $\boldsymbol{M}$ de task-uri este fezabil, atunci orice set concret de task-uri $\boldsymbol{W}$ generat din $\boldsymbol{M}$ poate fi planificat cu un anume algoritm non-preemptiv, şi în consecinţă, îndeplineşte relaţia (5-14).

Există însă cazuri în care seturi concrete de task-uri nu respectă **CN2** *şi totuşi pot fi planificate*. Modelul setului de ModX-uri independente propus în lucrare este echivalentul unui set concret de task-uri – anume setul pentru care momentele

primelor invocări sunt aliniate la $t = 0$: $\boldsymbol{W_0} = \{(M_1, 0), (M_2, 0), ..., (M_n, 0)\}$. Pentru acest model, **CN2** enunţată în [Jeffay 91] nu este aplicabilă, aşa cum o arată şi exemplul următor.

Exemplul 5-6.

Considerăm un set $\boldsymbol{M} = \{M_1, M_2, M_3, M_4\}$ de ModX-uri simple, independente, caracterizate prin următorii parametri temporali:

$$M_1 : \begin{cases} T_{pr}^{M_1} = 10 \\ T_{ex}^{M_1} = 4 \end{cases}; \quad M_2 : \begin{cases} T_{pr}^{M_2} = 15 \\ T_{ex}^{M_2} = 8 \end{cases}; \quad M_3 : \begin{cases} T_{pr}^{M_3} = 90 \\ T_{ex}^{M_3} = 4 \end{cases}; \quad M_4 : \begin{cases} T_{pr}^{M_4} = 90 \\ T_{ex}^{M_4} = 1 \end{cases}$$

În Figura 5-7 este reprezentată analiza offline a planificabilităţii setului $\boldsymbol{M}$ utilizând algoritmul *EDFNP* pentru intervalul $[0, T_{LCM})$, unde $T_{LCM} = 90$, cu rezultate pozitive. În consecinţă setul $\boldsymbol{M}$ este fezabil.

Din punct de vedere al condiţiilor de planificabilitate, setul $\boldsymbol{M}$ se conformează relaţiei (5-13) care interzice ca utilizarea procesor (egală cu 0.9888) să fie supraunitară, deci setul $\boldsymbol{M}$ se conformează condiţiei de necesitate **CN1**. În schimb, relaţia (5-14) nu este îndeplinită, existând un ModX (M_2) şi un interval temporal ($L = 11$) pentru care (5-14) e falsă:

$$L \geq T_{ex}^{M_i} + \sum_{j=1}^{i-1} \left\lfloor \frac{L-1}{T_{pr}^{M_j}} \right\rfloor \cdot T_{ex}^{M_j}, \text{ cu } i = 2, L = 11$$

$$\Rightarrow 11 \geq T_{ex}^{M_2} + \sum_{j=1}^{2-1} \left\lfloor \frac{L-1}{T_{pr}^{M_j}} \right\rfloor \cdot T_{ex}^{M_j}$$

$$\Rightarrow 11 \geq 8 + \left\lfloor \frac{11-1}{10} \right\rfloor \cdot 4 = 12 \quad \rightarrow \textbf{FALS!}$$

În consecinţă, condiţia **CN2** nu este îndeplinită, deşi setul $\boldsymbol{M}$ este totuşi fezabil.

❑

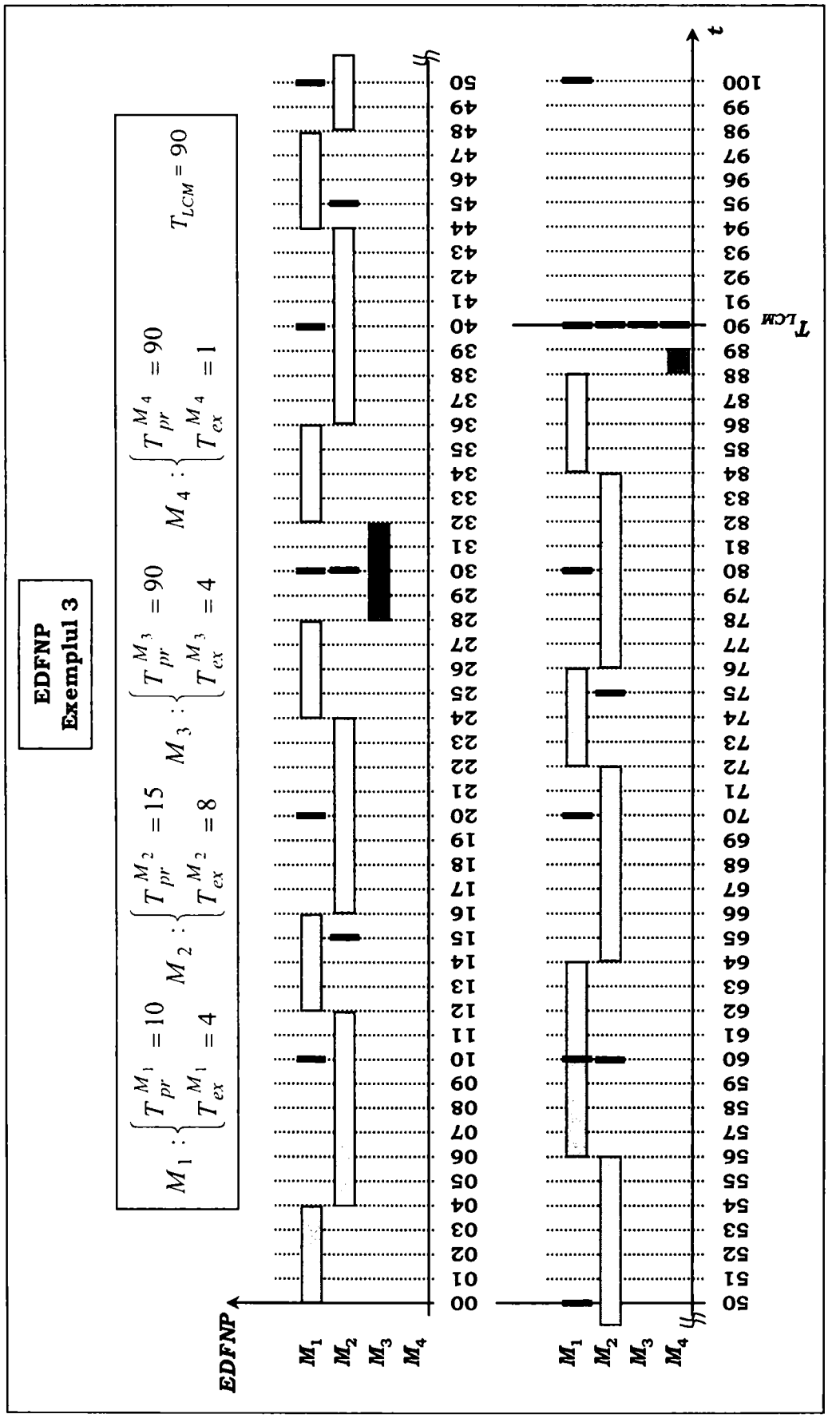

Figura 5-7. Planificarea EDFNP a setului M

Utilizarea ModX-urilor în dezvoltarea, analiza, planificarea şi execuţia sistemelor şi aplicaţiilor TR stricte necesită o condiţie suplimentară de necesitate a planificabilităţii. Această condiţie este impusă de modelul de task non-preemptiv pe care îl reprezintă ModX-ul.

Teorema 5-1. Condiţia a treia de necesitate a planificării ModX-urilor

Fie $\boldsymbol{M} = \{M_1, M_2, \ldots, M_n\}$, un set de n ModX-uri simple, independente, caracterizate din punct de vedere temporal prin durată de execuţie şi perioadă: $M_i \equiv \left(T_{ex}^{M_i}, T_{pr}^{M_i}\right)$, $i = 1..n$.

Dacă $\boldsymbol{M}$ este planificabil, atunci:

$$\textbf{CN3)} \qquad T_{ex}^{M_{ex\max}} \le 2\left(T_{pr}^{M_{pr\min}} - T_{ex}^{M_{pr\min}}\right) \tag{5-15}$$

unde $T_{ex}^{M_{ex\max}}$ reprezintă durata de execuţie a ModX-ului cu valoarea maximă pentru acest parametru din setul $\boldsymbol{M}$, iar $T_{pr}^{M_{pr\min}}$ şi $T_{ex}^{M_{pr\min}}$ reprezintă perioada, respectiv durata de execuţie a ModX-ului cu perioada minimă din setul $\boldsymbol{M}$.

Demonstraţie

Teorema impune practic o condiţie de limitare a duratei maxime de execuţie a oricărui ModX din $\boldsymbol{M}$, vis-a-vis de perioada minimă a ModX-urilor din set, şi se bazează pe faptul că ModX-urile nu pot fi întrerupte pe parcursul execuţiei acestora (operarea non-preemptivă). Figura 5-8 prezintă cazul cel mai defavorabil care poate să apară pentru parametrii temporali ai ModX-urilor unui set $\boldsymbol{M}$, care este totuşi fezabil.

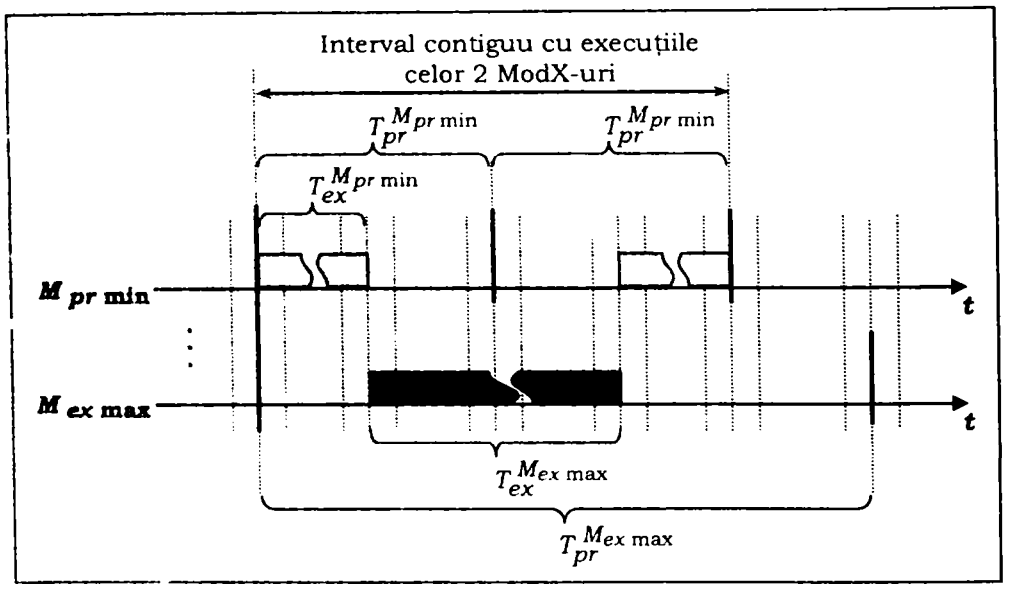

Figura 5-8. Cazul cel mai defavorabil pentru planificarea fezabilă a ModX-urilor unui set $\boldsymbol{M}$

În situaţia din Figura 5-8 sunt considerate cele două ModX-uri ale căror parametri temporali pot influenţa planificabilitatea din punctul de vedere al acestei teoreme. Se poate observa că intervalul de timp disponibil pentru planificarea lui $M_{ex\ max}$ astfel încât să nu se depăşească termenele ModX-urilor, este limitat de perioada lui $M_{pr\ min}$ şi de durata acestuia de execuţie, conform cu relaţia (5-15), care trebuia demonstrată.

❑

5.3 Planificarea online non-preemptivă a ModX-urilor independente

În subcapitolul precedent am tratat analiza offline a planificabilităţii non-preemptive a unui set de ModX-uri simple, independente, fără a lua însă în considerare modul cum poate fi implementată practic planificarea, pe un STR real. Rezultatele analizei offline constau (în cazul în care setul de ModX-uri ale aplicaţiei TR analizate este fezabil) dintr-o listă în care fiecare element e compus din indexul ModX-ului (identificatorul acestuia) şi momentul de timp la care e planificat. Astfel, pentru Exemplul 5-5, lista generată la analiza offline a planificabilităţii e prezentată în Tabelul 5-1, cu un total de 10 elemente, ce acoperă intervalul $[0, T_{LCM}) = [0, 40)$.

Din punct de vedere practic, al implementării planificării pe un STR real (*planificarea online*), se pune problema resurselor necesare, în speţă, a spaţiului de memorie necesar stocării în sistem a listei cu planificarea ModX-urilor. Dimensiunile listei de planificare variază puternic în funcţie de setul de ModX-uri care se doreşte a fi planificat şi de caracteristicile temporale ale acestora. De exemplu, în cazul în care setul conţine relativ multe ModX-uri şi cu perioade puternic diferite între ele (fără divizori comuni), iar duratele de execuţie sunt mici, rezultă în primul rând un interval de analiză T_{LCM} foarte mare, iar fiecare ModX din set va fi planificat de foarte multe ori în acest interval. Ca rezultat, spaţiul de memorie necesar planificării acestui set poate depăşi cu uşurinţă resursele disponibile ale sistemului.

Tabelul 5-1. Lista planificării EDFNP pentru Exemplul 5-5

Nr. element	Indice ModX	Moment planificare
1	1	0
2	2	3
3	1	9
4	2	12
5	1	18
6	2	21
7	1	27
8	2	30
9	1	36
10	3	39

Modelul de sistem TR strict propus în lucrarea de faţă presupune existenţa în memoria sistem a unei tabele de dimensiune bine stabilită şi a unui task TR strict (un ModX special) cu rolul de a completa în mod ciclic tabela cu ModX-urile rezultate în urma algoritmului de planificare implementat.

Tabela ce conţine ModX-urile planificate împreună cu momentele de start ale execuţiei acestora, şi pe care o denumim *Tabela de Dispatch*, este de fapt un buffer, accesat pe de o parte de către *online scheduler* M_S, pentru a o completa ciclic, şi pe de altă parte, de către executivul sistemului, *Dispatcher*, cu rolul de lansare în execuţie a câte unui ModX din tabelă, la momentul specificat de planificare. Notăm cu λ *dimensiunea utilă* a Tabelei de Dispatch, adică numărul elemente ale tabelei ce conţin ModX-uri ale aplicaţiei, planificate de către M_S în cadrul unui *ciclu de planificare*.

În continuare vom introduce şi discuta două abordări diferite ce vizează implementarea practică a algoritmilor de planificare a seturilor de ModX-uri în timpul operării STR (planificarea online).

5.3.1 Planificarea online în cicluri cu număr constant de execuţii

Planificarea online în cicluri cu număr constant de execuţii, *CEC-NPOS* (*Constant Execution Cycles – Non-Preemptive Online Scheduling*) presupune ca pentru fiecare ciclu de planificare să existe un acelaşi număr de planificări de ModX-uri. Prin această metodă se asigură utilizarea la maxim a capacităţii de memorie alocată pentru Tabela de Dispatch.

Abordarea planificării în cicluri cu număr constant de execuţii presupune o structură de buffer circular pentru Tabela de Dispatch (Figura 5-9), în care prima poziţie este ocupată întotdeauna de execuţia viitoare a planificatorului M_S (execuţia ce defineşte următorul ciclu de planificare), iar restul de λ locaţii corespund planificărilor de ModX-uri sistem şi aplicaţie din cadrul ciclului curent. Atunci când executivul STR termină ultima execuţie planificată din Tabela de Dispatch (M_k, planificat la momentul t_λ în cazul din figură), se iniţiază un nou ciclu de planificare. Noul ciclu începe cu execuţia planificatorului M_S, parcurgerea Tabelei de Dispatch fiind deci reluată de la prima înregistrare.

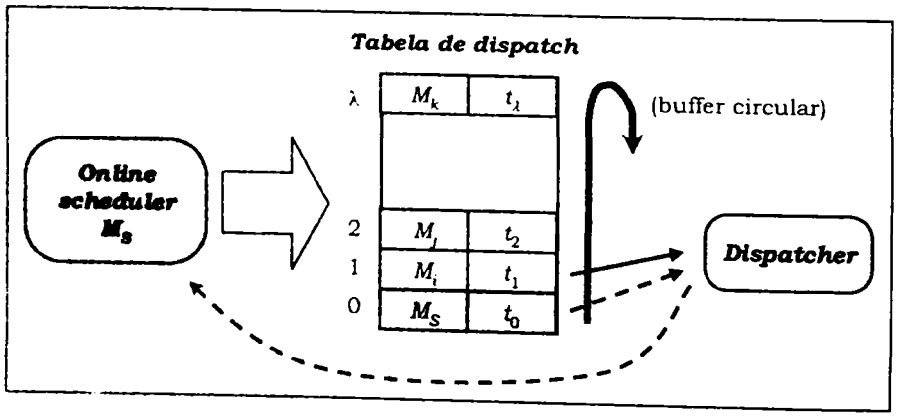

Figura 5-9. Structura de principiu a Tabelei de Dispatch

Task-ul M_S (*online scheduler*) este un caz special de ModX, cu următoarele caracteristici definitorii:

- Execuţia lui M_S se desfăşoară tot context non-preemptiv;
- $T_{ex}^{M_S}$ este durata de execuţie;
- Perioada lui M_S, $T_{pr}^{M_S}$, nu se defineşte. Aşadar, execuţia M_S nu este periodică în timp, în schimb task-ul prezintă o execuţie ce se repetă ciclic de fiecare dată ce au fost executate λ planificări de ModX-uri ale aplicaţiei;
- Algoritmul de planificare implementat de către M_S va fi o versiune similară a algoritmului utilizat în analiza offline a planificării aplicaţiei, pentru a avea garanţia că în timpul operării sistemul se comportă absolut în conformitate cu analiza efectuată offline.

În acest context, analiza offline a planificării unui set ***M*** de ModX-uri simple, independente, va trebui să includă şi ModX-ul special M_S, care nu este periodic, ci trebuie planificat ciclic, după fiecare λ planificări normale de ModX-uri din ***M***. Algoritmul de planificare propus este o versiune a lui *EDFNP*, modificată corespunzător. A rezultat algoritmul denumit *CEC-NPOS*, cu organigrama operării de principiu prezentată în Figura 5-10. Algoritmul *CEC-NPOS* planifică prima dată (la momentul $t = 0$) pe task-ul M_S, avansează timpul de planificare cu $t = t + T_{ex}^{M_S}$, după care aplică *EDFNP* pentru următoarele λ planificări de ModX-uri ale aplicaţiei (incrementând un contor intern de planificări). În continuare, este planificat din nou M_S, şi aşa mai departe până se ajunge la acoperirea intervalului de planificare, T_{LCM}.

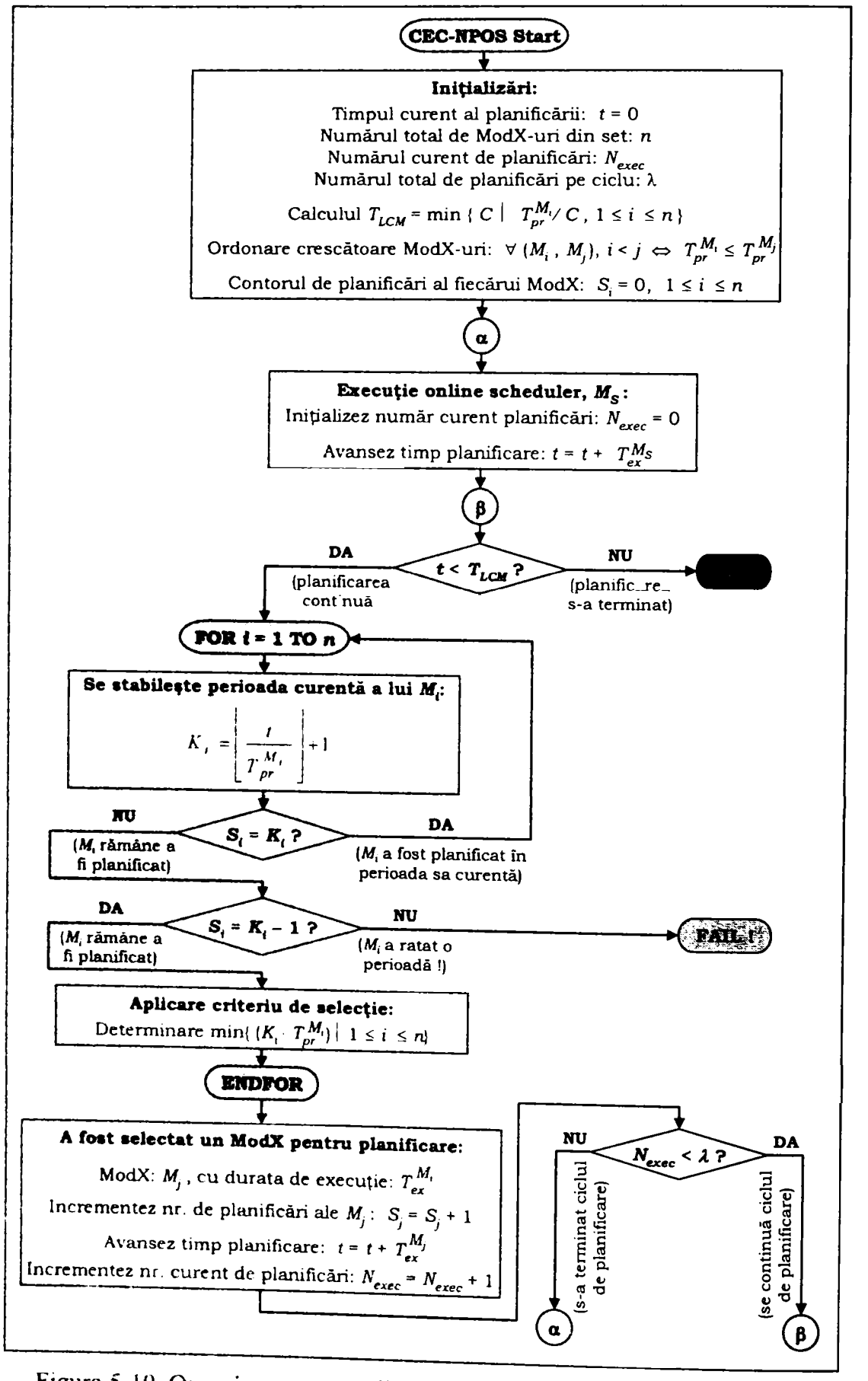

Figura 5-10. Organigrama operării de principiu a algoritmului *CEC-NPOS*

Introducerea ModX-ului special M_S, care nu e periodic în timp ci are o execuţie ciclică, în număr de planificări ale celorlalte ModX-uri din setul $\boldsymbol{M}$ supus analizei, modifică şi condiţiile de planificabilitate discutate în secţiunea anterioară. Introducem următoarea teoremă:

Teorema 5-2. Condiţia de necesitate a planificării *CEC-NPOS*

Fie $\boldsymbol{M} = \{M_1, M_2, \ldots, M_n\}$, un set de n ModX-uri simple, independente, caracterizate din punct de vedere temporal prin durată de execuţie şi perioadă: $M_i \equiv \left(T_{ex}^{M_i}, T_{pr}^{M_i}\right)$. ModX-urile din $\boldsymbol{M}$ sunt ordonate crescător după perioadă (pentru orice pereche (M_i, M_j), dacă $i < j$ atunci $T_{pr}^{M_i} \le T_{pr}^{M_j}$). Considerăm un sistem TR ţintă ce dispune de o Tabelă de Dispatch de dimensiune $\lambda + 1$ şi de un ModX special de planificare online M_S, cu durata de execuţie $T_{ex}^{M_S}$.

Dacă setul $\boldsymbol{M}$ este planificabil cu algoritmul *CEC-NPOS*, atunci:

$$\sum_{i=1}^{n} \frac{T_{ex}^{M_i}}{T_{pr}^{M_i}} + \left\lceil \frac{T_{LCM} \cdot \sum_{i=1}^{n} \frac{1}{T_{pr}^{M_i}}}{\lambda} \right\rceil \cdot \frac{T_{ex}^{M_S}}{T_{LCM}} \le 1 \qquad (5\text{-}16)$$

Demonstraţie

Dacă setul M este planificabil, rezultă că planificarea cu *CEC-NPOS* pe intervalul T_{LCM} va avea rezultate pozitive.

Notăm cu N_S, numărul total de execuţii ale ModX-ului special M_S în intervalul de lungime T_{LCM}. Dar, N_S va fi egal cu numărul total de execuţii ale ModX-urilor din $\boldsymbol{M}$ pe durata T_{LCM}, raportat la λ:

$$N_S = \left\lceil \frac{T_{LCM} \cdot \sum_{i=1}^{n} \frac{1}{T_{pr}^{M_i}}}{\lambda} \right\rceil \qquad (5\text{-}17)$$

Prin urmare, fracţiunea din timpul procesor ocupată cu execuţia lui M_S pe durata intervalului T_{LCM} este:

$$U^{M_S}(T_{LCM}) = \frac{N_S}{T_{LCM}} \cdot T_{ex}^{M_S} = \left\lceil \frac{T_{LCM} \cdot \sum_{i=1}^{n} \frac{1}{T_{pr}^{M_i}}}{\lambda} \right\rceil \cdot \frac{T_{ex}^{M_S}}{T_{LCM}} \qquad (5\text{-}18)$$

Pe de altă parte, condiția planificării pe sisteme monoprocesor specifică faptul că suma fracțiunilor din timpul procesor ocupate de execuția task-urilor planificate trebuie să nu fie supraunitară, pentru orice interval de timp. Rezultă, pentru cazul nostru:

$$U^{\boldsymbol{M}+M_S}(\infty) = U^{\boldsymbol{M}+M_S}(T_{LCM}) = U^{\boldsymbol{M}}(T_{LCM}) + U^{M_S}(T_{LCM}) = \\ = \sum_{i=1}^{n} \frac{T_{ex}^{M_i}}{T_{pr}^{M_i}} + U^{M_S}(T_{LCM}) \le 1 \qquad (5\text{-}19)$$

Înlocuind pe (5-18) în (5-19), obținem condiția de necesitate a planificării *CEC-NPOS* specificată de (5-16).

❑

Exemplul 5-7.

Considerăm un set $\boldsymbol{M} = \{M_1, M_2, M_3, M_4\}$ de ModX-uri simple, independente, caracterizate prin următorii parametri temporali:

$$M_1 : \begin{cases} T_{pr}^{M_1} = 10 \\ T_{ex}^{M_1} = 2 \end{cases}; \quad M_2 : \begin{cases} T_{pr}^{M_2} = 15 \\ T_{ex}^{M_2} = 4 \end{cases}; \quad M_3 : \begin{cases} T_{pr}^{M_3} = 20 \\ T_{ex}^{M_3} = 4 \end{cases}; \quad M_4 : \begin{cases} T_{pr}^{M_4} = 90 \\ T_{ex}^{M_4} = 4 \end{cases}$$

Setul $\boldsymbol{M}$ urmează a fi planificat în cadrul unui STR cu ajutorul planificatorului online M_S, ce dispune de o Tabelă de Dispatch de dimensiune 10 + 1 (deci $\lambda = 10$).

Înainte de a începe analiza offline a planificabilității setului $\boldsymbol{M}$ (pentru intervalul $[0, T_{LCM})$, unde $T_{LCM} = 180$), se va efectua verificarea condiției de necesitate a planificabilității, enunțată în Teorema 5-2. Utilizarea procesor ce corespunde setului $\boldsymbol{M}$ de ModX-uri are valoarea:

$$U^{\boldsymbol{M}}(T_{LCM}) = U^{\boldsymbol{M}}(180) = \sum_{i=1}^{4} \frac{T_{ex}^{M_i}}{T_{pr}^{M_i}} = 0.711$$

Pe de altă parte, utilizarea procesor ce corespunde execuțiilor M_S pe intervalul $[0, T_{LCM}) = [0, 180)$, se determină aplicând (5-18):

$$U^{M_S}(T_{LCM}) = U^{M_S}(180) = \left\lceil \frac{180 \cdot \sum_{i=1}^{4} \frac{1}{T_{pr}^{M_i}}}{10} \right\rceil \cdot \frac{7}{180} = 0.194$$

Utilizarea procesor a întregului sistem de task-uri, compus din setul $\boldsymbol{M}$ împreună cu ModX-ul special M_S, se determină ca fiind:

$$U^{\boldsymbol{M}+M_S}(180) = U^{\boldsymbol{M}}(180) + U^{M_S}(180) = 0.905 \qquad (5\text{-}20)$$

Din (5-20) rezultă că suma fracțiunilor din timpul procesor ocupate de execuția task-urilor planificate nu este supraunitară, deci setul ***M*** îndeplineşte condiția impusă de planificarea *CEC-NPOS* pe sistemul TR stabilit ca țintă.

În Figura 5-11 este reprezentată analiza offline a planificabilității setului ***M*** utilizând algoritmul *CEC-NPOS* pentru intervalul $[0, T_{LCM})$, unde $T_{LCM} = 180$, cu rezultate pozitive. În consecință setul ***M*** este fezabil din punct de vedere al planificării şi execuției pe sistemul țintă.

❑

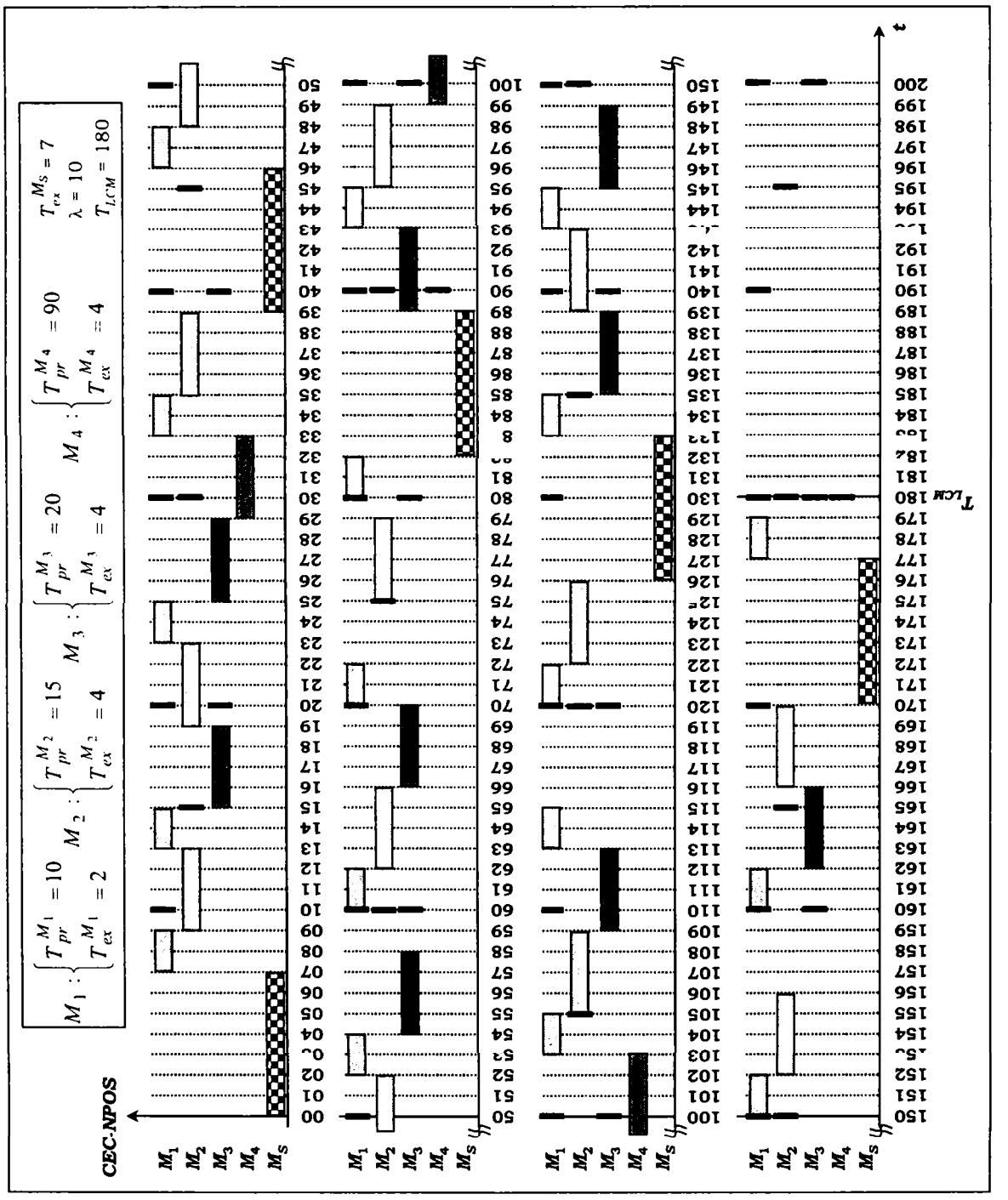

Figura 5-11. Analiza offline a planificabilității *CEC-NPOS* a setului ***M***

5.3.2 Planificarea online în cicluri periodice

Metoda planificării online, în cicluri cu număr constant de execuţii, descrisă anterior, impune o serie de restricţii pentru setul de ModX-uri sistem şi ale aplicaţiei, în principal din cauză că task-ul de planificare M_S nu respectă modelul definit pentru ModX-uri (nu este un task periodic). Mai mult, aşa cum se va vedea în subcapitolul următor, în cazul în care setul de ModX-uri supus planificării conţine şi ModX-uri cu execuţie fixă în cadrul perioadei acestora (aşa-numitele FModX-uri), abordarea planificării *CEC-NPOS* nu mai este fezabilă.

Rezolvarea acestor probleme pentru planificarea online poate fi găsită cu următoarele soluţii:

- Existenţa în sistem a unui procesor dedicat planificărilor online: aşa-numitul "scheduling coprocessor", propus pentru arhitectura sistemului TR distribuit "Spring" [Stankovic 91a, Stankovic 91b, Niehaus 93, Stankovic 99]. În această situaţie, sistemul ţintă devine un sistem multiprocesor, cu un procesor destinat în exclusivitate planificărilor, şi unul sau mai multe procesoare pentru execuţia task-urilor aplicaţiei. Această soluţie oferă în primul rând o importantă creştere a eficienţei întregului sistem, degrevându-l de încărcarea suplimentară impusă de execuţia planificatorului, care, din punctul de vedere al aplicaţiilor, este complet transparentă. Datorită acestei perspective, considerăm abordarea ca un subiect important pentru cercetarea viitoare din domeniu. Ca dezavantaj, se poate totuşi menţiona creşterea în complexitate a sistemului şi algoritmilor de planificare, ce trebuie să vizeze sistemele multiprocesor şi cele distribuite.
- Având în vedere o configuraţie monoprocesor a sistemului ţintă (subiectul preocupărilor curente de cercetare), rezolvarea problemei o poate constitui introducerea unui planificator online periodic, care respectă în totalitate caracteristicile modelului de ModX. De aici rezultă abordarea *PC-NPOS* (*Periodic Cycles – Non-Preemptive Online Scheduling*).

Faptul că M_S este un ModX, deci un task periodic, executat în context non-preemptiv, relaxează restricţiile de planificare impuse celorlaltor ModX-uri din set, dar, pe de altă parte complică problema utilizării Tabelei de Dispatch, datorită faptului că numărul total de înregistrări utilizate diferă de la un ciclu de planificare la altul. Se pune problema determinării valorii perioadei lui M_S, $T_{pr}^{M_S}$, astfel încât capacitatea tabelei să nu fie depăşită pentru nici un ciclu de planificare, pe toată durata operării sistemului.

Teorema 5-3. A planificării online periodice (*PC-NPOS*)

Considerăm o Tabelă de Dispatch de dimensiune totală λ +1 înregistrări (similară ca structură a înregistrărilor cu cea prezentată în Figura 5-9). ModX-ul de planificare online este definit prin următorii parametri temporali: $M_S \equiv \left(T_{pr}^{M_S}, T_{ex}^{M_S} \right)$. Considerăm de asemenea $\boldsymbol{M} = \{M_1, M_2, \ldots, M_n\}$, un set de n ModX-uri ordonate crescător după perioadă:

$$M_i \equiv \left(T_{pr}^{M_i}, T_{ex}^{M_i} \right) \text{ şi } \forall \left(M_i, M_j \right). i < j \Leftrightarrow T_{pr}^{M_i} \leq T_{pr}^{M_j}$$

Pentru a se putea realiza planificarea online a setului ***M*** împreună cu ModX-ul M_S, fără a se depăşi capacitatea totală a Tabelei de Dispatch, este necesară verificarea relaţiei:

$$\sum_{i=1}^{n} \left\lceil \frac{2\left(T_{pr}^{M_S} - T_{ex}^{M_S} \right)}{T_{pr}^{M_i}} \right\rceil \leq \lambda \qquad (5\text{-}21)$$

Demonstraţie

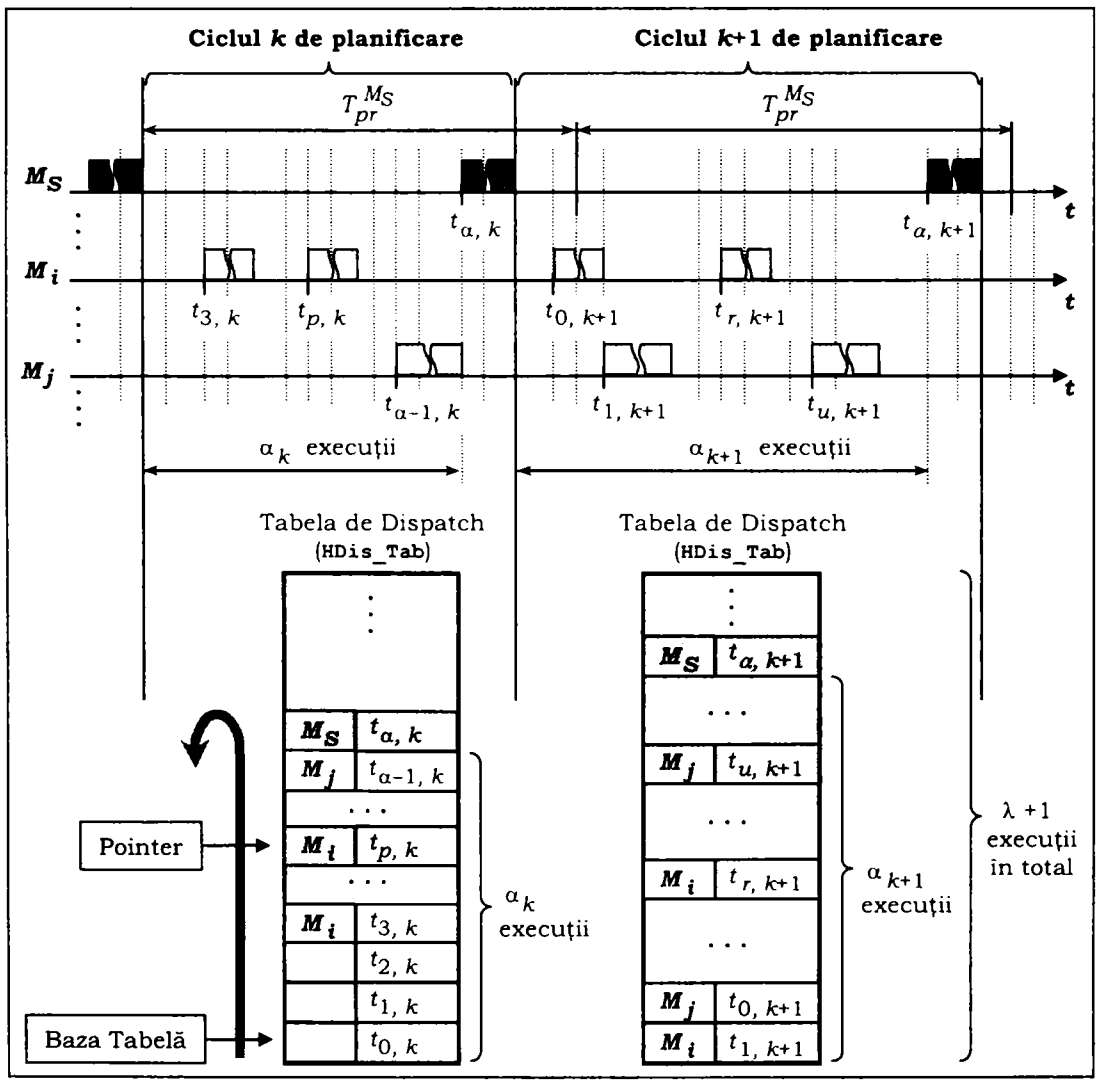

Figura 5-12. Exemplificare a două cicluri consecutive de planificare

Figura 5-12 exemplifică execuţia M_S şi a ModX-urilor din $\boldsymbol{M}$ pentru două cicluri consecutive de planificare, k şi $k + 1$. Structura Tabelei de Dispatch pentru cele două cicluri este de asemenea ilustrată în figură. Se poate observa cum numărul de înregistrări de descriu planificările variază de la un ciclu la altul. Ciclurile de planificare sunt periodice, fiind definite de două execuţii consecutive ale lui M_S (practic, un ciclu se termină cu planificarea M_S).

Pentru a nu se depăşi capacitatea totală a Tabelei de Dispatch, trebuie ca numărul de planificări ale ModX-urilor din $\boldsymbol{M}$ pentru oricare ciclu, să respecte condiţia: $\alpha_k \leq \lambda$, sau:

$$\alpha \leq \lambda \ , \text{ unde: } \alpha = \max\left\{\alpha_k \,\middle|\, k = 1,2,...\right\} \tag{5-22}$$

Parametrul α_k depinde în mod direct de intervalul de timp din cadrul ciclului k de planificare: cu cât L_k este mai lung, cu atât α_k poate fi mai mare. Cel mai defavorabil caz din acest punct de vedere este cel în care M_S se execută cu întârzierea cea mai mare posibilă, faţă de execuţia precedentă (Figura 5-13).

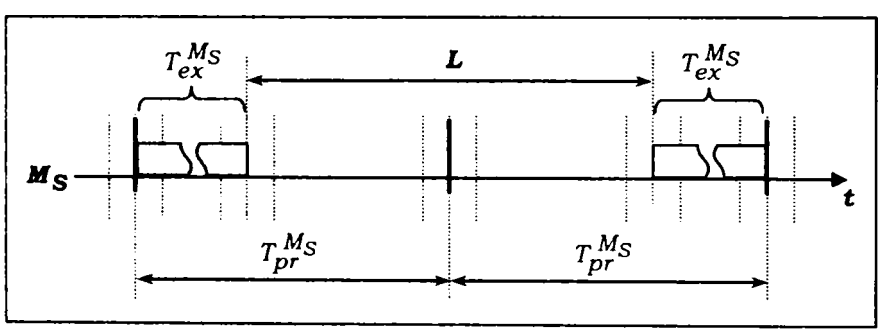

Figura 5-13. Intervalul cel mai lung dintre două execuţii succesive ale M_S

$$L = \max\left\{L_k \,\middle|\, k = 1,2,...\right\} = 2\left(T_{pr}^{M_S} - T_{ex}^{M_S}\right) \tag{5-23}$$

Numărul maxim de execuţii ale ModX-urilor din M ce pot fi planificate într-un interval de lungime (durată) L este:

$$\alpha = \sum_{i=1}^{n} \left\lceil \frac{L}{T_{pr}^{M_i}} \right\rceil \tag{5-24}$$

Din (5-22), (5-23) şi (5-24) rezultă relaţia (5-21), care exprimă restricţiile impuse celor trei parametri care definesc planificarea online periodică pe un sistem ţintă: perioada şi durata de execuţie a planificatorului M_S, şi dimensiunea maximă (utilă) a Tabelei de Dispatch.

❑

Considerând parametrii $T_{ex}^{M_S}$ şi λ cu valori fixate/calculate apriori, pe baza relaţiei (5-21) se poate determina o perioadă pentru M_S ($T_{pr}^{M_S}$), suficient de mică pentru a nu se depăşi dimensiunea totală a Tabelei de Dispatch, dar în acelaşi timp, convenabil de mare pentru a asigura o performanţă de operare a sistemului cât mai bună posibil. Termenul însumat în (5-21) este limitat, conform definiţiei operatorului "⌈ ⌉" ("ceiling"), astfel:

$$2\frac{T_{pr}^{M_S} - T_{pr}^{M_S}}{T_{pr}^{M_i}} \le \left\lceil 2\frac{T_{pr}^{M_S} - T_{pr}^{M_S}}{T_{pr}^{M_i}} \right\rceil \le 2\frac{T_{pr}^{M_S} - T_{pr}^{M_S}}{T_{pr}^{M_i}} + 1 \qquad (5\text{-}25)$$

Prin urmare, o valoare minimală dar validă, pentru perioada M_S se poate obţine din (5-21) şi (5-25):

$$\sum_{i=1}^{n}\left(2\frac{\left(T_{pr}^{M_S}\right)_{\min} - T_{ex}^{M_S}}{T_{pr}^{M_i}} + 1 \right) \le \lambda$$

$$2\left(\left(T_{pr}^{M_S}\right)_{\min} - T_{ex}^{M_S}\right)\sum_{i=1}^{n}\frac{1}{T_{pr}^{M_i}} + n \le \lambda$$

$$\Rightarrow \quad \left(T_{pr}^{M_S}\right)_{\min} = \left\lfloor \frac{\lambda - n}{2\sum_{i=1}^{n}\frac{1}{T_{pr}^{M_i}}} \right\rfloor + T_{ex}^{M_S} \qquad (5\text{-}26)$$

De la valoarea $\left(T_{pr}^{M_S}\right)_{\min}$ obţinută cu (5-26), se poate determina un maxim valid pentru perioada M_S, prin incrementări succesive, astfel încât noua valoare să respecte relaţia (5-21). Se poate uşor observa că această metodă de determinare a perioadei planificatorului online M_S atunci când i se cunoaşte durata de execuţie şi dimensiunea maximă a Tabelei de Dispatch, se pretează la o abordare algoritmică, automată, putând fi implementată în contextul unor unelte soft de verificare şi analiză automată a planificabilităţii setului de ModX-uri ce compune o aplicaţie TR oarecare.

Dimensiunea utilă a Tabelei de Dispatch, λ, este supusă la rândul ei unor restricţii de limită inferioară. Pentru ca relaţia (5-26) să poată fi evaluată, este necesar ca $\lambda - n > 0$, adică:

$$\lambda > n \qquad (5\text{-}27)$$

Limita inferioară a lui λ depinde aşadar în mod direct de numărul total de ModX-uri din $\boldsymbol{M}$. Cu cât n este mai mare, cu atât este necesară o Tabelă de Dispatch mai mare.

Pe de altă parte, deoarece şi M_S este, din punct de vedere al sistemului, tot un ModX, discuţiile legate de condiţiile de planificabilitate **CN1)**, **CN2)** şi **CN3)** din subcapitolul anterior, rămân valabile şi se aplică şi pentru setul extins $\boldsymbol{M} \cup \{M_S\}$. Astfel, de exemplu durata de execuţie a planificatorului online trebuie să fie suficient de mică pentru a satisface **CN3)**:

$$T_{ex}^{M_S} \le 2\left(T_{pr}^{M_{pr\min}} - T_{ex}^{M_{pr\min}}\right) \tag{5-28}$$

În concluzie, introducerea planificatorului online periodic şi a Tabelei de Dispatch în analiza operării sistemului ţintă, impune restricţiile (5-21), (5-27) şi (5-28) pentru parametrii care definesc acest tip de planificare: $T_{pr}^{M_S}$, $T_{ex}^{M_S}$ şi λ.

Exemplul 5-8.

Considerăm un set $\boldsymbol{M} = \{M_1, M_2, M_3, M_4\}$ de ModX-uri simple, independente, caracterizate prin următorii parametri temporali:

$$M_1:\begin{cases} T_{pr}^{M_1} = 10 \\ T_{ex}^{M_1} = 1 \end{cases};\quad M_2:\begin{cases} T_{pr}^{M_2} = 15 \\ T_{ex}^{M_2} = 7 \end{cases};\quad M_3:\begin{cases} T_{pr}^{M_3} = 24 \\ T_{ex}^{M_3} = 2 \end{cases};\quad M_4:\begin{cases} T_{pr}^{M_4} = 24 \\ T_{ex}^{M_4} = 1 \end{cases}$$

Setul $\boldsymbol{M}$ urmează a fi planificat (metoda *PC-NPOS*) în cadrul unui STR cu ajutorul planificatorului online periodic M_S, ce dispune de o Tabelă de Dispatch de dimensiune 12 + 1 (deci $\lambda = 12$). Considerăm că durata de execuţie a lui M_S este $T_{ex}^{M_S} = 8$.

Prima relaţie care se verifică este (5-27): $\lambda = 12 > n = 4$. ModX-ul cu cea mai mică perioadă din set este M_1, astfel încât şi (5-28) este verificată:

$$T_{ex}^{M_S} \le 2\left(T_{pr}^{M_1} - T_{ex}^{M_1}\right) \Leftrightarrow 8 \le 2(10-1) = 18$$

În continuare, se determină perioada minimă a lui M_S, cu ajutorul (5-26):

$$\left(T_{pr}^{M_S}\right)_{\min} = \left\lfloor (12-4) \Big/ \left(2\sum_{i=1}^{n}\frac{1}{T_{pr}^{M_i}}\right)\right\rfloor + 8 = 24$$

De la această valoare, se determină perioada efectivă a lui M_S, prin incrementare succesivă şi verificarea relaţiei (5-21). În final, se va ajunge la $T_{pr}^{M_S} = 30$.

Figura 5-14 prezintă diagrama de analiză offline a planificabilităţii *PC-NPOS* a setului extins $\boldsymbol{M} \cup \{M_S\}$. Încărcarea procesor implicată de ModX-urile din $\boldsymbol{M}$ este: $U^{\boldsymbol{M}}(120) = 0.691$, iar cea a setului extins: $U^{\boldsymbol{M}+M_S}(120) = 0.958$.

❑

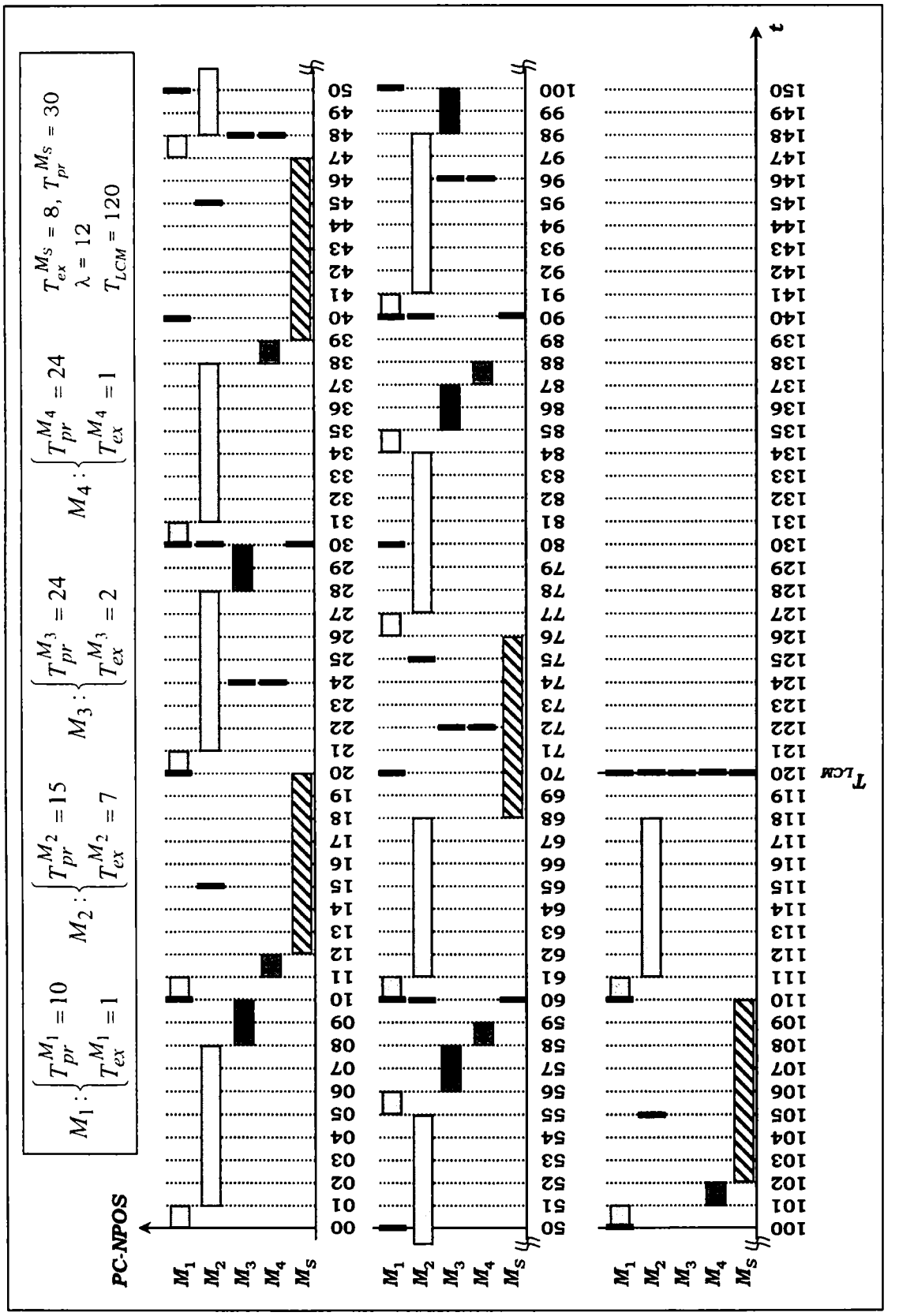

Figura 5-14. Analiza offline a planificabilității *PC-NPOS* a setului ***M***

5.4 Planificarea non-preemptivă a ModX-urilor independente cu execuţie fixă

5.4.1 Introducere

În subcapitolele anterioare a fost tratată problema planificării non-preemptive a unui set de ModX-uri simple, independente, pe sisteme TR stricte. Algoritmii de planificare discutaţi, dintre care *EDFNP* a fost găsit ca cel mai eficient, sunt capabili a planifica seturi fezabile de ModX-uri, astfel încât acestea să-şi respecte termenele stricte impuse de specificaţiile de comportare temporală.

Aşa cum rezultă din exemplele 5-2, 5-4 până la 5-7, respectiv figurile 5-3, 5-5 până la 5-7, 5-10 şi 5-14 momentul exact de planificare a fiecărui ModX în cadrul perioadei sale variază de la o perioadă la alta (de la un ciclu de execuţie la altul). Există însă o clasă importantă de aplicaţii TR stricte, şi deci un număr relativ mare de cazuri de ModX-uri, pentru care momentul execuţiei în cadrul fiecărei perioade trebuie să rămână constant de-a lungul operării. Exemple de aplicaţii cu astfel de cerinţe pentru unele ModX-uri includ achiziţiile periodice de date, generatoarele de semnale/de funcţii programabile, sistemele de comunicaţie sincrone, etc.

Exemplul 5-9.

Considerăm o aplicaţie TR de generare a unui semnal de ieşire strict periodic, de o anumită formă. Pentru simplificare, presupunem ca necesară doar condiţia ca primul eşantion al semnalului dintr-o perioadă oarecare să fie egal distanţat în timp de eşantionul corespunzător al perioadelor vecine, pe toată durata generării semnalului.

Task-ul care generează semnalul (sau doar primul eşantion al fiecărei perioade a semnalului) trebuie să fie periodic, şi, mai mult, intervalele de timp ce separă începutul fiecărei execuţii a task-ului trebuie să fie egale cu perioada semnalului (vezi Figura 5-15).

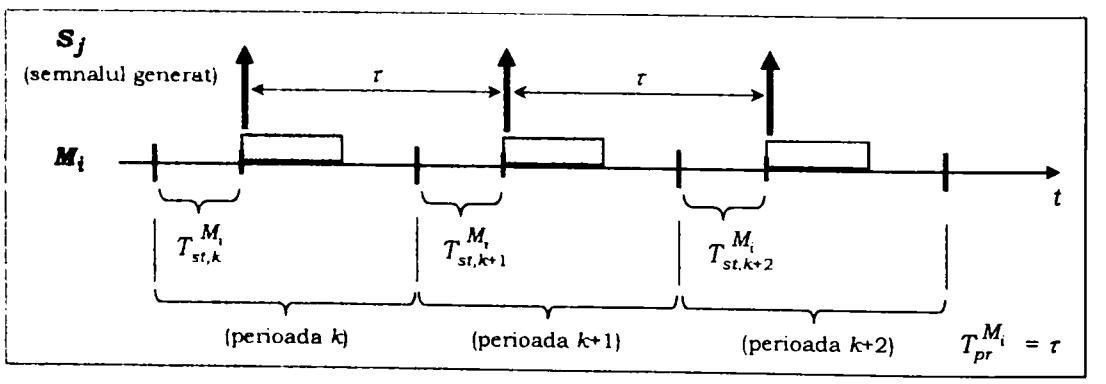

Figura 5-15. Generarea unui semnal de ieşire perfect periodic

Cea mai simplă abordare constă din utilizarea mecanismului de întreruperi şi programarea unui timer pentru perioada τ. Problema acestei abordări constă însă în cazurile în care apar alte întreruperi mai prioritare în sistem, adică există alte task-uri, mai prioritare, ce întrerup execuţia lui M_i, compromiţând astfel generarea semnalului la perioade egale.

Utilizând modelul de ModX-uri, propus în lucrarea de faţă, problema întreruperilor este rezolvată. ModX-ul M_i, responsabil pentru generarea semnalului S_j, va avea perioada $T_{pr}^{M_i} = \tau$. Va mai trebui prevăzută însă o restricţie în faza de specificare, anume aceea că M_i se va executa în mod obligatoriu cu aceeaşi întârziere faţă de momentul de început al fiecărei perioade, adică:

$$T_{st,k}^{M_i} = T_{st,k+1}^{M_i} = T_{st}^{M_i} \quad , \forall k = 1,2,\ldots \tag{5-29}$$

❑

Definiţia 5-3.

Numim *ModX cu execuţie fixă*, sau mai simplu, *FModX*, un ModX a cărui planificare şi execuţie se vor efectua în mod obligatoriu cu o aceeaşi întârziere faţă de momentul de început al fiecărei perioade (5-29). Cu alte cuvinte, intervalul dintre oricare două execuţii efective ale unui FModX în sistem este constant şi egal chiar cu perioada FModX-ului. Intervalul de timp dintre startul unei perioade oarecare a unui FModX şi momentul execuţiei sale este denumit *interval* (sau *durată*) *de start*, şi e notat cu $T_{st}^{M_i}$.

❑

Ca observaţie, menţionăm că specificarea unui ModX cu execuţie fixă se poate realiza şi cu ajutorul modelului de ModX descris în capitolul precedent, utilizând cei doi parametri care restricţionează momentul execuţiei în cadrul perioadei – intervalul de întârziere ($T_{dy}^{M_i}$) şi intervalul limită ($T_{dl}^{M_i}$):

$$\begin{cases} T_{dy}^{M_i} = T_{st}^{M_i} \\ T_{dl}^{M_i} = T_{st}^{M_i} + T_{ex}^{M_i} \end{cases}$$

În paragrafele următoare vom aborda problema planificării non-preemptive a ModX-urilor cu execuţie fixă, independente (*FENP*: *Fixed Execution Non-Preemptive*). Vom propune şi demonstra un model matematic pentru FModX-uri, pe care îl vom utiliza la studiul planificării non-preemptive a seturilor de FModX-uri independente.

5.4.2 Modelul matematic al ModX-urilor cu execuţie fixă

Definiţia 5-4.
Definim funcţia $M_i(t)$ după cum urmează:

$$M_i(t): \mathbf{N} \to \{0,1\}, \text{ cu :}$$
$$M_i(t) = \sigma\left(t \bmod T_{pr}^{M_i} - T_{st}^{M_i}\right) - \sigma\left(t \bmod T_{pr}^{M_i} - T_{st}^{M_i} - T_{ex}^{M_i}\right) \quad (5\text{-}30)$$

unde: "mod" reprezintă *operatorul modulo*, iar $\sigma : \mathbf{Z} \to \{0,1\}$, $\sigma(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases}$

este *funcţia treaptă unitate*.

Dacă stabilim convenţia ca valoarea "1" a funcţiei $M_i(t)$ reprezintă starea de execuţie a unui ModX M_i la momentul t, atunci funcţia $M_i(t)$ modelează din punct de vedere matematic comportarea în timp a operării ModX-urilor cu execuţie fixă (a FModX-urilor).

❑

Pentru ilustrarea definiţiei modelului matematic al FModX-urilor, prezentăm următorul exemplu:

Exemplul 5-10.
Considerăm un ModX cu execuţie fixă M_1, caracterizat prin următorii parametri temporali (Figura 5-16):

$$M_1 : \begin{cases} T_{pr}^{M_1} = 8 \\ T_{ex}^{M_1} = 3 \\ T_{st}^{M_1} = 2 \end{cases}$$

Operarea FModX-ului M_1 este modelată de funcţia:

$$M_1(t) = \sigma(t \bmod 8 - 2) - \sigma(t \bmod 8 - 2 - 3) = \sigma(t \bmod 8 - 2) - \sigma(t \bmod 8 - 5)$$

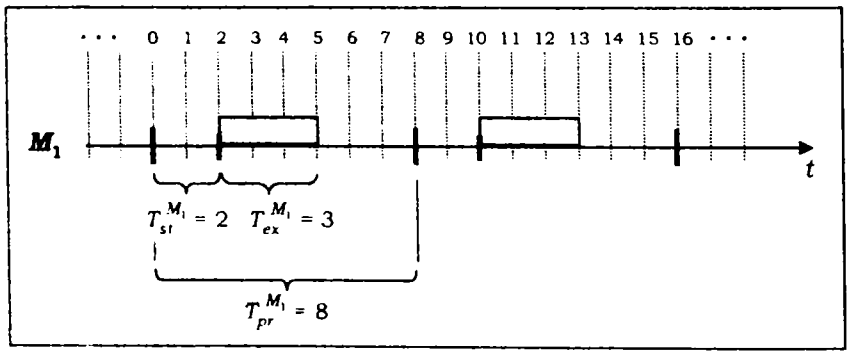

Figura 5-16. Operarea FModX-ului M1 pentru (primele) două perioade ale sale

Valorile funcției $M_1(t)$ pentru câteva instanțe de timp sunt după cum urmează:

$$M_1(1) = \sigma(1 \bmod 8 - 2) - \sigma(1 \bmod 8 - 5) = \sigma(-1) - \sigma(-4) = 0$$
$$M_1(3) = \sigma(3 \bmod 8 - 2) - \sigma(3 \bmod 8 - 5) = \sigma(1) - \sigma(-2) = 1$$
$$M_1(5) = \sigma(5 \bmod 8 - 2) - \sigma(5 \bmod 8 - 5) = \sigma(3) - \sigma(0) = 0$$
$$M_1(14) = \sigma(14 \bmod 8 - 2) - \sigma(14 \bmod 8 - 5) = \sigma(4) - \sigma(1) = 0$$

❑

Modelul matematic al ModX-urilor cu execuție fixă se bazează pe operatorul "mod" şi pe proprietățile "aritmeticii modulare" din teoria numerelor [Chouinard 02, Ning 02, Lerma 03]. Câteva proprietăți ale operatorului modulo, care vor fi utilizate în studiul planificării FModX-urilor, sunt enunțate în continuare.

Fie $a, b, c \in Z$ şi $n \in Z^*$.

Proprietatea 5-1. Operatorul (mod n) împarte Z în n seturi de întregi:
$\forall\, a \in Z,\ a \in Z_n = \{0, 1, \ldots, (n-1)\}$.

Proprietatea 5-2. Comutativitatea adunării şi înmulțirii modulare:
$$\begin{cases} (a+b) \bmod n = (b+a) \bmod n \\ (a \cdot b) \bmod n = (b \cdot a) \bmod n \end{cases}, \quad \forall a, b \in \mathbf{Z}_n .$$

Proprietatea 5-3. Asociativitatea:
$$\begin{cases} ((a+b)+c) \bmod n = (a+(b+c)) \bmod n \\ ((a \cdot b) \cdot c) \bmod n = (a \cdot (b \cdot c)) \bmod n \end{cases}, \quad \forall a, b, c \in \mathbf{Z}_n .$$

Proprietatea 5-4. Distributivitatea:
$(a \cdot (b + c)) \bmod n = (a \cdot b + a \cdot c) \bmod n$, $\forall\, a, b, c \in Z_n$.

Proprietatea 5-5. Elementul neutru:
$$\begin{cases} (0+a) \bmod n = a \bmod n \\ (1 \cdot a) \bmod n = a \bmod n \end{cases}, \quad \forall a \in \mathbf{Z}_n .$$

Proprietatea 5-6. Elementul invers față de adunarea modulară:
$\forall\, a \in Z_n, \exists\, b \in Z_n\ ;\ (a + b) \bmod n = 0$.

Proprietatea 5-7. Elementul invers față de înmulțirea modulară:
$\forall\, a \in Z_n, \exists\, b \in Z_n\ ;\ (a \cdot b) \bmod n = 1$,
dacă şi numai dacă a şi n sunt relativ prime (cel mai mare divizor comun al lui a şi n este 1).

Proprietatea 5-8. Descompunerea modulară faţă de adunare şi înmulţire:

$$\begin{cases}(a+b)\bmod n=(a\bmod n+b\bmod n)\bmod n\\(a\cdot b)\bmod n=(a\bmod n\cdot b\bmod n)\bmod n\end{cases},\ \forall a,b\in\mathbf{Z}_n.$$

Proprietatea 5-9. Congruenţa a două numere:

$\forall\ a, b \in Z_n$; $\left(a \overset{\text{not}}{\equiv} b\right) \bmod n$, dacă şi numai dacă n divide pe $(a-b)$ (adică, $n/(a-b)$), sau $(a-b) \bmod n = 0$.

5.4.3 Planificarea unui set de două FModX-uri

Fie $\boldsymbol{M} = \{M_1, M_2\}$, un set de două ModX-uri independente, cu execuţie fixă, ordonate crescător după perioadă:

$$\begin{cases}M_1\equiv\left(T_{pr}^{M_1},T_{ex}^{M_1},T_{st}^{M_1}\right)\\M_2\equiv\left(T_{pr}^{M_2},T_{ex}^{M_2},T_{st}^{M_2}\right)\end{cases},\ \text{cu } T_{pr}^{M_1}\le T_{pr}^{M_2}.$$

Conform cu Definiţia 5-4, cele două FModX-uri sunt modelate cu ajutorul următoarelor funcţii:

$$\begin{cases}M_1(t)=\sigma\left(t\bmod T_{pr}^{M_1}-T_{st}^{M_1}\right)-\sigma\left(t\bmod T_{pr}^{M_1}-T_{st}^{M_1}-T_{ex}^{M_1}\right)\\M_2(t)=\sigma\left(t\bmod T_{pr}^{M_2}-T_{st}^{M_2}\right)-\sigma\left(t\bmod T_{pr}^{M_2}-T_{st}^{M_2}-T_{ex}^{M_2}\right)\end{cases},\ t\in\mathbf{N}$$

Pentru studiul planificării non-preemptive a setului M, considerăm următoarele notaţii:

- Cel mai mare divizor comun al perioadelor lui M_1 şi M_2:

$$\mathcal{D}\overset{\text{not}}{=}\text{GCD}\left\{T_{pr}^{M_1},T_{pr}^{M_2}\right\}=\max\left\{\delta\in\mathbf{N}^*\,\middle|\,T_{pr}^{M_1}\bmod\delta=T_{pr}^{M_2}\bmod\delta=0\right\}\quad(5\text{-}31)$$

- Cel mai mic multiplu comun al perioadelor lui M_1 şi M_2:

$$\mathcal{M}\overset{\text{not}}{=}\text{LCM}\left\{T_{pr}^{M_1},T_{pr}^{M_2}\right\}=\min\left\{\mu\in\mathbf{N}^*\,\middle|\,\mu\bmod T_{pr}^{M_1}=\mu\bmod T_{pr}^{M_2}=0\right\}\quad(5\text{-}32)$$

Definiția 5-5.

Definim *maparea planificării fixe a lui* M_1 *în perioada lui* M_2 $\left(T_{pr}^{M_2}\right)$, ca fiind o funcție de forma:

$$\Delta_{M_2/M_1} : \left\{0,1,\ldots,T_{pr}^{M_2}-1\right\} \to \{0,1\}$$

$$\Delta_{M_2/M_1}(\tau) = \bigcup_{k=0}^{\frac{T_{pr}^{M_1}}{\mathcal{D}}-1} M_1\left(\tau + k \cdot T_{pr}^{M_2}\right) \tag{5-33}$$

❑

Funcția de mapare a planificării fixe a lui M_1 în perioada lui M_2 calculează pentru orice moment de timp τ relativ la intervalul $T_{pr}^{M_2}$ o valoare ce specifică dacă M_1 se află în execuție sau nu, de-a lungul operării setului $\boldsymbol{M}$ în cadrul STR. Calculul unei valori $\Delta_{M_2/M_1}(\tau)$ se bazează pe reunirea (operatorul "$\cup$" – "sau" logic) valorilor funcției $M_1(t)$ pentru toate instanțele de timp t care se mapează în "poziția relativă" τ în cadrul perioadei lui M_2, de-a lungul operării setului $\boldsymbol{M}$. Exemplul următor ilustrează un caz practic de calcul a funcției de mapare $\Delta_{M_2/M_1}(\tau)$.

Exemplul 5-11.

Considerăm un set $\boldsymbol{M} = \{M_1, M_2\}$, compus din două FModX-uri independente, cu următoarele caracteristici temporale:

$$M_1 : \begin{cases} T_{pr}^{M_1} = 15 \\ T_{ex}^{M_1} = 3 \\ T_{st}^{M_1} = 0 \end{cases} ; \quad M_2 : \begin{cases} T_{pr}^{M_2} = 20 \\ T_{ex}^{M_2} = 1 \\ T_{st}^{M_2} = 4 \end{cases}$$

Cei doi parametri suplimentari ce derivă din caracteristicile temporale ale setului $\boldsymbol{M}$ sunt, conform (5-31) şi (5-32):

$$\begin{cases} \mathcal{D} = \mathrm{GCD}\left\{T_{pr}^{M_1}, T_{pr}^{M_2}\right\} = \mathrm{GCD}\{15,20\} = 5 \\ \mathcal{M} = \mathrm{LCM}\left\{T_{pr}^{M_1}, T_{pr}^{M_2}\right\} = \mathrm{LCM}\{15,20\} = 60 \end{cases}$$

adică, cel mai mare divizor comun, respectiv cel mai mic multiplu comun al perioadelor lui M_1 şi M_2. Conform relației (5-33), funcția de mapare a planificării fixe a lui M_1 în perioada lui M_2 este de forma:

$$\Delta_{M_2/M_1}(\tau) = \bigcup_{k=0}^{2} M_1(\tau + k \cdot 20), \text{ cu } \tau = 0,1,\ldots,19 .$$

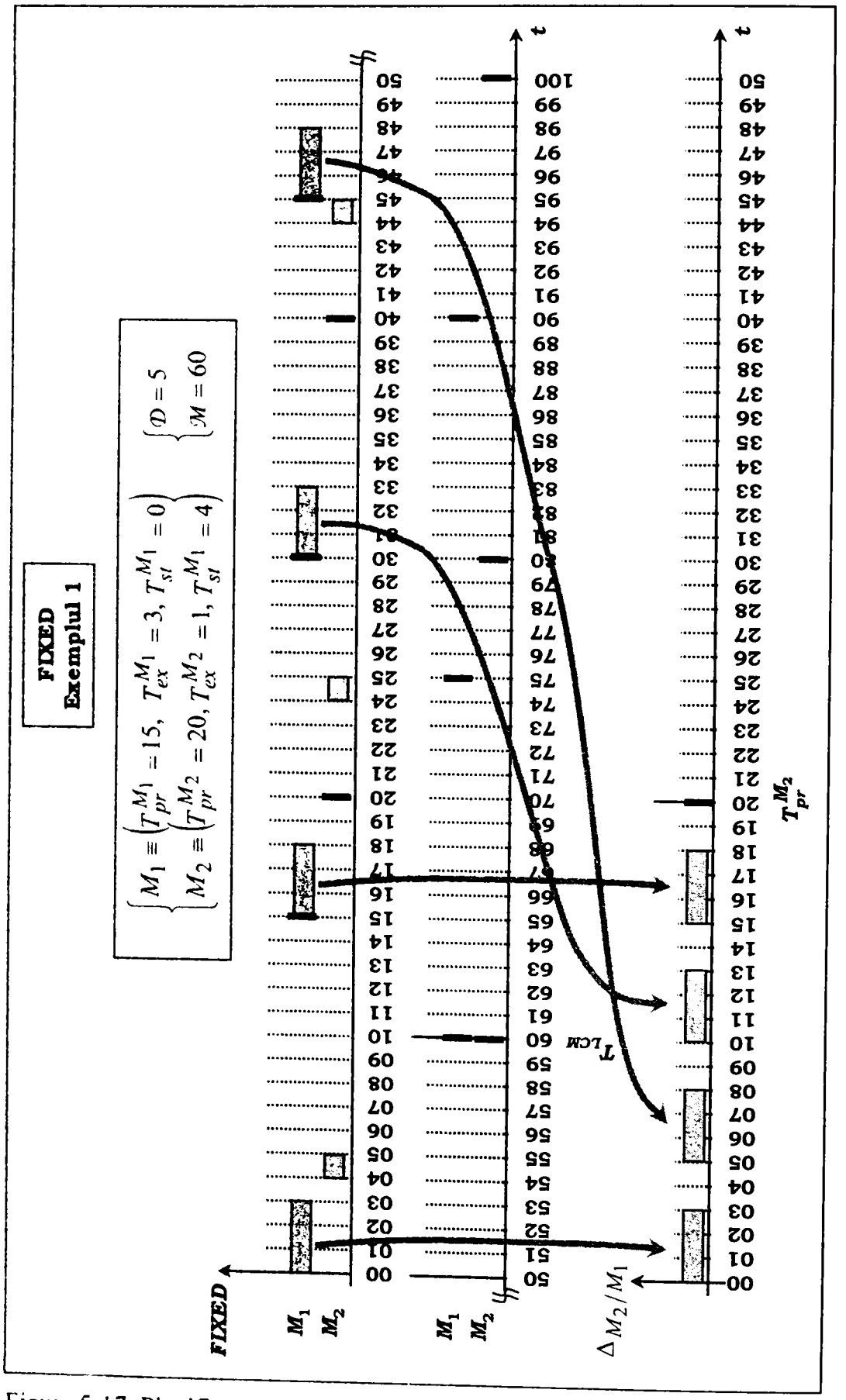

Figura 5-17. Planificarea fixă a setului M de FModX-uri şi funcţia de mapare

Încărcarea procesor a setului ***M*** nu este supraunitară, satisfăcând astfel condiția de necesitate **CN1)** a planificabilității (5-13):

$$U^{\boldsymbol{M}} = \sum_{i=1}^{2} \frac{T_{ex}^{M_i}}{T_{pr}^{M_i}} = 0.25 \le 1$$

Figura 5-17 ilustrează analiza offline a planificabilității setului ***M*** de FModX-uri şi funcția de mapare $\Delta_{M_2/M_1}(\tau)$. Calculul funcției de mapare pentru instanțele timpului relativ la perioada lui M_2, τ, se efectuează utilizând relațiile (5-33) şi (5-30):

$$\Delta_{M_2/M_1}(1) = \bigcup_{k=0}^{2} M_1(1+k\cdot 20) = M_1(1) \cup M_1(21) \cup M_1(41) =$$
$$= (\sigma(1 \bmod 15 - 0) - \sigma(1 \bmod 15 - 3)) \cup$$
$$\cup (\sigma(21 \bmod 15 - 0) - \sigma(21 \bmod 15 - 3)) \cup$$
$$\cup (\sigma(41 \bmod 15 - 0) - \sigma(41 \bmod 15 - 3)) =$$
$$= (\sigma(1) - \sigma(-2)) \cup (\sigma(6) - \sigma(3)) \cup (\sigma(11) - \sigma(8)) = 1 \cup 0 \cup 0 = 1$$

$$\Delta_{M_2/M_1}(9) = \bigcup_{k=0}^{2} M_1(9+k\cdot 20) = M_1(9) \cup M_1(29) \cup M_1(49) =$$
$$= (\sigma(9 \bmod 15 - 0) - \sigma(9 \bmod 15 - 3)) \cup$$
$$\cup (\sigma(29 \bmod 15 - 0) - \sigma(29 \bmod 15 - 3)) \cup$$
$$\cup (\sigma(49 \bmod 15 - 0) - \sigma(49 \bmod 15 - 3)) =$$
$$= (\sigma(9) - \sigma(6)) \cup (\sigma(14) - \sigma(11)) \cup (\sigma(4) - \sigma(1)) = 0 \cup 0 \cup 0 = 0$$

❑

Înainte de a aborda planificarea unui set de ModX-uri independente cu execuție fixă utilizând funcția de mapare $\Delta_{M_2/M_1}(\tau)$, vom studia câteva proprietăți importante ale acesteia.

Teorema 5-4.

Fie $\boldsymbol{M} = \{M_1, M_2\}$, un set de două ModX-uri cu execuţie fixă, ordonate crescător după perioadă:

$$\begin{cases} M_1 \equiv \left(T_{pr}^{M_1}, T_{ex}^{M_1}, T_{st}^{M_1}\right) \\ M_2 \equiv \left(T_{pr}^{M_2}, T_{ex}^{M_2}, T_{st}^{M_2}\right) \end{cases}, \text{ cu } T_{pr}^{M_1} \le T_{pr}^{M_2},$$

şi $\mathcal{D} \in \mathbf{N}^*$, cel mai mare divizor comun al perioadelor lui M_1 şi M_2:

$$\mathcal{D} \overset{\text{not}}{=} \text{GCD}\left\{T_{pr}^{M_1}, T_{pr}^{M_2}\right\} = \max\left\{\delta \in \mathbf{N}^* \,\middle|\, T_{pr}^{M_1} \bmod \delta = T_{pr}^{M_2} \bmod \delta = 0\right\}.$$

Funţia de mapare a planificării fixe a lui M_1 în perioada lui M_2, $\Delta_{M_2/M_1}(\tau)$, este periodică, de perioadă $\mathcal{D}$:

$$\Delta_{M_2/M_1}(t_0) = \Delta_{M_2/M_1}(t_0 + \mathcal{D}), \ \forall t_0 \in \left\{0,1,\ldots,T_{pr}^{M_2} - 1\right\} \tag{5-34}$$

Demonstraţie

Din definiţia funcţiei de mapare (5-33) şi din (5-34), va trebui demonstat că:

$$\begin{gathered} \forall k \in \left\{0,1,\ldots,\frac{T_{pr}^{M_1}}{\mathcal{D}} - 1\right\}, \ \exists m \in \left\{0,1,\ldots,\frac{T_{pr}^{M_1}}{\mathcal{D}} - 1\right\}, \text{ astfel incat :} \\ M_1\left(t_0 + k \cdot T_{pr}^{M_2}\right) = M_1\left(t_0 + \mathcal{D} + m \cdot T_{pr}^{M_2}\right) \end{gathered} \tag{5-35}$$

Introducem notaţiile:

$$\begin{cases} T_{pr}^{M_1} = \alpha \cdot \mathcal{D} \\ T_{pr}^{M_2} = \beta \cdot \mathcal{D} \end{cases}, \text{ cu } T_{pr}^{M_1} \le T_{pr}^{M_2},$$

de unde rezultă că $\alpha \le \beta$, şi mai departe:

$$\beta \overset{\text{not}}{=} \gamma \cdot \alpha + \delta \tag{5-36}$$

unde: $\delta \in Z_\alpha = \{0, 1, \ldots, \alpha - 1\}$, iar α şi δ sunt relativ prime (GCD{α, δ} = 1). Relaţia (5-36) se obţine din faptul că $\text{GCD}\left\{T_{pr}^{M_1}, T_{pr}^{M_2}\right\} = \mathcal{D}$. Presupunând că GCD{$\alpha$, δ} = ξ > 1, ar rezulta că $\text{GCD}\left\{T_{pr}^{M_1}, T_{pr}^{M_2}\right\} = \mathcal{D} \cdot \xi$, fals.

Cu notaţiile de mai sus şi aplicând definiţia funcţiei $M_1(t)$ conform (5-30), formula (5-35) ce trebuie demonstrată devine:

$$\begin{gathered} \forall k \in \mathbf{Z}_\alpha = \{0,1,\ldots,\alpha - 1\}, \ \exists m \in \mathbf{Z}_\alpha, \text{ astfel incat :} \\ (t_0 + k \cdot \beta \cdot \mathcal{D}) \bmod (\alpha \cdot \mathcal{D}) = (t_0 + (m \cdot \beta + 1) \cdot \mathcal{D}) \bmod (\alpha \cdot \mathcal{D}) \end{gathered} \tag{5-37}$$

Aplicând Proprietatea 5-9 a operatorului "mod", (5-37) se transformă în:

$$(t_0 + k \cdot \beta \cdot \mathcal{D} - t_0 - \mathcal{D} - m \cdot \beta \cdot \mathcal{D}) \bmod (\alpha \cdot \mathcal{D}) = 0$$

Utilizând descompunerea modulară (Proprietatea 5-8), rezultă:

$$((k-m) \cdot \beta) \bmod \alpha = 1$$
$$((k-m) \bmod \alpha \cdot (\gamma \cdot \alpha + \delta) \bmod \alpha) \bmod \alpha = 1$$
$$((k-m) \bmod \alpha \cdot \delta) \bmod \alpha = 1$$

Dar, (5-36) permite existenţa elementului invers faţă de înmulţirea modulară (Proprietatea 5-7):

$$\exists \mu \in \mathbf{Z}_\alpha \text{, astfel incat } (\mu \cdot \delta) \bmod \alpha = 1$$

Rezultă:

$$(k-m) \bmod \alpha = \mu \text{, sau}$$

$$m = (k-\mu) \bmod \alpha \qquad (5\text{-}38)$$

Deci, pentru orice $k \in Z_\alpha$, $\exists\, m \in Z_\alpha$, cu $m = (k-\mu) \bmod \alpha$ şi $\mu \in Z_\alpha$, astfel încât formula (5-35) este adevărată. Mai mult, m ia toate valorile din mulţimea $Z_\alpha = \{0, 1, \ldots, \alpha - 1\}$, pe măsură ce şi k baleiază întreagul Z_α.

❑

Corolarul 5-1.

Funcţia de mapare a planificării fixe a lui M_1 în perioada lui M_2, $\Delta_{M_2/M_1}(\tau)$, conţine exact ν_{M_2/M_1} perioade, unde:

$$\nu_{M_2/M_1} = \frac{T_{pr}^{M_2}}{\mathcal{D}} \qquad (5\text{-}39)$$

Demonstraţie

Demonstraţia este o consecinţă imediată a definiţiei funcţiei $\Delta_{M_2/M_1}(\tau)$ (Definiţia 5-5) şi a Teoremei 5-4. Astfel, ştim că $\Delta_{M_2/M_1}(\tau)$ este definită pentru $\tau \in \{0,1,\ldots,T_{pr}^{M_2} - 1\}$, adică pentru $T_{pr}^{M_2}$ unităţi de timp, şi este o funcţie periodică, de perioadă D. Rezultă că

$$\nu_{M_2/M_1} = \frac{T_{pr}^{M_2}}{\mathcal{D}} = \beta$$

reprezintă numărul de perioade ale lui $\Delta_{M_2/M_1}(\tau)$, ce împart un interval de durată $T_{pr}^{M_2}$.

❑

Ca observaţie, menţionăm că pe durata unui interval de planificare ($T_{LCM} = \text{LCM}\{T_{pr}^{M_1}, T_{pr}^{M_2}\} = \mathcal{M}$, conform relaţiei (5-32)), se vor mapa în perioada lui M_2 exact

$$\frac{LCM\{T_{pr}^{M_1}, T_{pr}^{M_2}\}}{T_{pr}^{M_1}} = \frac{T_{pr}^{M_2}}{\mathcal{D}} = \nu_{M_2/M_1}$$

execuţii ale lui M_1. De aici şi din Corolarul 5-1 rezultă că funcţia de mapare a planificării fixe a lui M_1 în perioada lui M_2 este o funcţie definită pe $T_{pr}^{M_2}$ instanţe de timp, pe care le împarte în ν_{M_2/M_1} intervale identice (perioade), de lungime $\mathcal{D}$, iar fiecare perioadă a lui $\Delta_{M_2/M_1}(\tau)$ include (mapează) o execuţie a lui M_1, de durată $T_{ex}^{M_1}$.

Cu ajutorul funcţiei de mapare şi a proprietăţilor acesteia, studiate mai sus, putem aborda problema planificării non-preemptive a unui set oarecare de două ModX-uri cu execuţie fixă.

Teorema 5-5.

Fie $\boldsymbol{M} = \{M_1, M_2\}$, un set de două ModX-uri cu execuţie fixă, ordonate crescător după perioadă:

$$\begin{cases} M_1 \equiv \left(T_{pr}^{M_1}, T_{ex}^{M_1}, T_{st}^{M_1}\right) \\ M_2 \equiv \left(T_{pr}^{M_2}, T_{ex}^{M_2}, T_{st}^{M_2}\right) \end{cases}, \text{ cu } T_{pr}^{M_1} \le T_{pr}^{M_2}.$$

Setul $\boldsymbol{M}$ este planificabil în context non-preemptiv, dacă şi numai dacă:

$$\exists t_0 \in \{0,1,\ldots,T_{pr}^{M_2} - T_{ex}^{M_2}\}, \text{ astfel incat } \bigcup_{\tau=t_0}^{t_0+T_{ex}^{M_2}-1} \Delta_{M_2/M_1}(\tau) = 0 \qquad (5\text{-}40)$$

Demonstraţie

a) Implicaţia directă: dacă (5-40) e adevărată, rezultă că setul $\boldsymbol{M}$ este fezabil.

Dacă (5-40) este adevărată, rezultă că în cadrul perioadei lui M_2 există un interval de timp de durată egală cu $T_{ex}^{M_2}$, care începe de la momentul t_0 şi în cadrul căruia nu se mapează execuţia lui M_1, pentru toată durata planificării. De aici rezultă faptul că intervalul de timp $[t_0, t_0 + T_{ex}^{M_2} - 1]$ poate fi alocat execuţiei lui M_2 în cadrul planificării setului $\boldsymbol{M}$.

Deoarece execuţiile lui M_1 şi M_2 nu se vor suprapune pe durata planificării în condiţiile de mai sus, rezultă că există o planificare fezabilă a setului $\boldsymbol{M}$.

b) Implicaţia inversă: dacă setul $\boldsymbol{M}$ este fezabil, rezultă că (5-40) se verifică.

Dacă setul $\boldsymbol{M}$ este fezabil, rezultă că există o planificare a lui M_1 şi M_2 astfel încât execuţiile acestora să nu se suprapună pe durata operării.

Acest lucru este echivalent cu faptul că maparea execuției lui M_1 în perioada lui M_2 pe durata planificării, conține cel puțin un interval de timp care este mai mare sau egal cu durata de execuție a lui M_2. De aici rezultă că (5-40) este adevărată.

❑

Corolarul 5-2.

Dacă setul $\boldsymbol{M} = \{M_1, M_2\}$ de FModX-uri este planificabil, astfel încât

$$\exists\, t_0 \in \left\{0,1,\ldots,T_{pr}^{M_2} - T_{ex}^{M_2}\right\},\ \text{cu} \bigcup_{\tau=t_0}^{t_0+T_{ex}^{M_2}-1} \Delta_{M_2/M_1}(\tau) = 0,$$

atunci: $T_{st}^{M_2} = t_0$.

Demonstrație

Demonstrația corolarului derivă direct din demonstrația Teoremei 5-5.

❑

Corolarul 5-3.

Dacă setul $\boldsymbol{M} = \{M_1, M_2\}$ de FModX-uri este planificabil, atunci:

$$T_{ex}^{M_1} + T_{ex}^{M_2} \le \mathcal{D} = \text{GCD}\left\{T_{pr}^{M_1}, T_{pr}^{M_2}\right\} \tag{5-41}$$

Demonstrație

Teorema 5-5 afirmă că pentru ca setul $\boldsymbol{M}$ să fie fezabil, trebuie să existe cel puțin un interval de valori succesive ale lui τ, pentru care funcția de mapare $\Delta_{M_2/M_1}(\tau)$ este nulă, iar intervalul respectiv va avea o lungime (durată) cel puțin egală cu $T_{ex}^{M_2}$. Notăm acest interval cu T_x:

$$T_x \ge T_{ex}^{M_2} \tag{5-42}$$

Pe de altă parte, conform Teoremei 5-4, funcția de mapare este periodică, de perioadă D. Prin urmare, T_x nu poate fi mai mare decât perioada funcției $\Delta_{M_2/M_1}(\tau)$: $T_x \le$ D.

Conform aceleiaşi teoreme şi a Corolarului 5-1, în cadrul fiecărei perioade a funcției $\Delta_{M_2/M_1}(\tau)$ este mapată câte o execuție a lui M_1. Cu alte cuvinte, în cadrul fiecărei perioade D, funcția de mapare are valoarea "1" pentru $T_{ex}^{M_1}$ instanțe consecutive ale lui τ. Rezultă că:

$$T_x \le \mathcal{D} - T_{ex}^{M_1} \tag{5-43}$$

Din (5-42) şi (5-43) rezultă că relația (5-41) din enunț se verifică.

❑

Corolarul 5-3 exprimă practic, *condiţiile de necesitate şi suficienţă* ale planificării non-preemptive pentru un set de două (şi numai două!) ModX-uri independente, cu execuţie fixă (*FENP*):

> *"Condiţia necesară şi suficientă pentru ca un set de două ModX-uri independente, cu execuţie fixă, să poată fi planificat în context non-preemptiv este ca suma duratelor lor de execuţie să nu depăşească valoarea celui mai mare divizor comun al perioadelor acestora."*

În cazul în care setul de FModX-uri îndeplineşte condiţiile de necesitate şi suficienţă, se poate avansa la stabilirea unei planificări fezabile, constând practic din determinarea intervalelor de start $T_{st}^{M_i}$ ale FModX-urilor, pe baza funcţiei de mapare $\Delta_{M_2/M_1}(\tau)$.

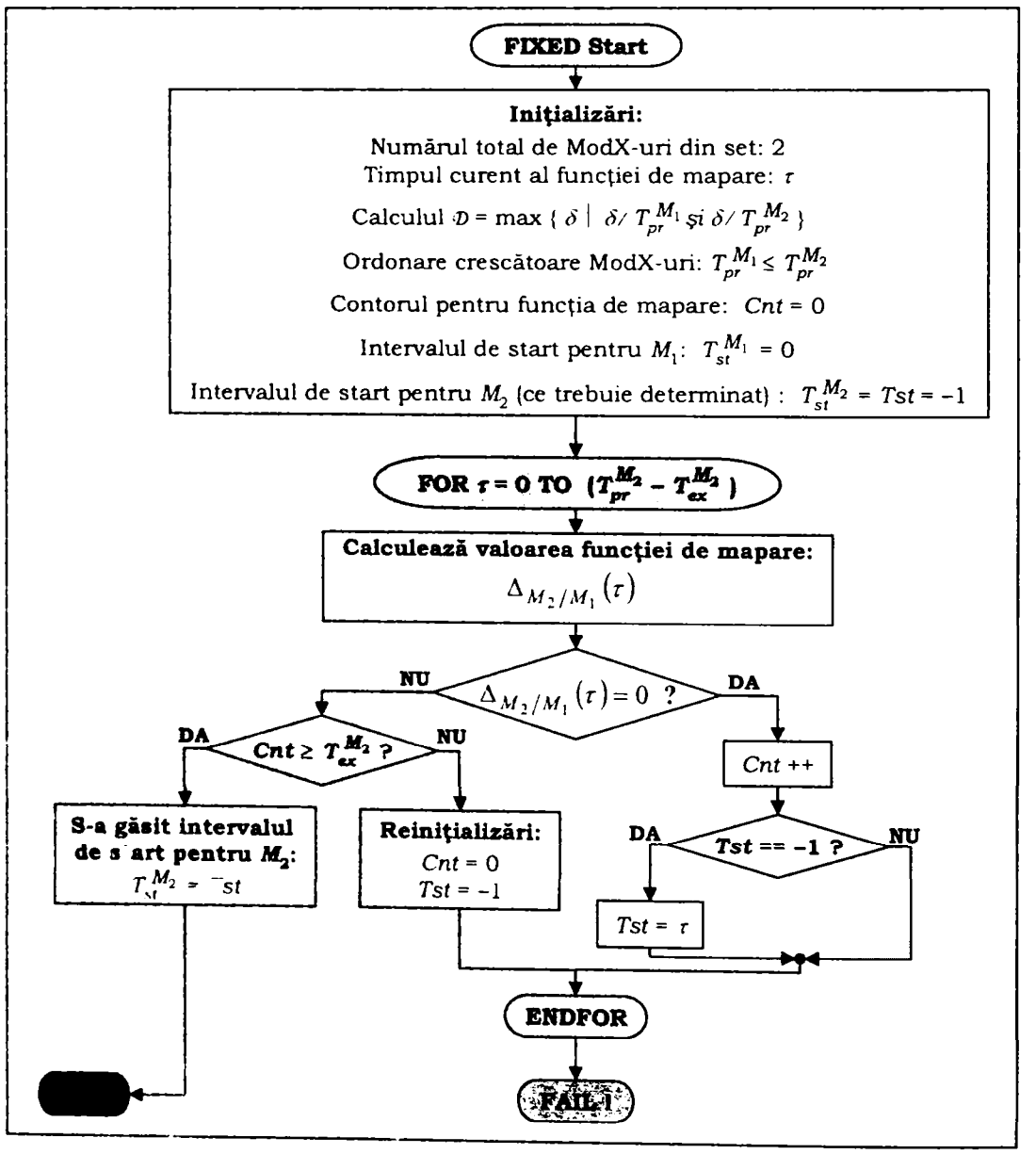

Figura 5-18. Algoritmul de planificare non-preemptivă a unui set de două FModX-uri

Algoritmul de planificare non-preemptivă a două FModX-uri independente (Figura 5-18), porneşte de la ipoteza că intervalul de start al lui M_1 este cunoscut apriori (în cazul nostru, pentru simplitate, $T_{st}^{M_1} = 0$) şi urmăreşte determinarea lui $T_{st}^{M_2}$. Pentru aceasta, utilizează o variabilă care contorizează numărul de valori nule consecutive ale funcției de mapare $\Delta_{M_2/M_1}(\tau)$.

În cazul setului de FModX-uri din Exemplul 5-11, algoritmul va obține $T_{st}^{M_2} = 3$, rezultând într-o planificare fezabilă a setului (alta decât cea ilustrată în Figura 5-17). Exemplul următor prezintă toate procedurile implicate în analiza offline a planificabilității setului de FModX-uri considerat.

Exemplul 5-12.

Considerăm un set $\boldsymbol{M} = \{M_1, M_2\}$, compus din două FModX-uri independente, cu următoarele caracteristici temporale:

$$M_1 : \begin{cases} T_{pr}^{M_1} = 30 \\ T_{ex}^{M_1} = 5 \end{cases}; \quad M_2 : \begin{cases} T_{pr}^{M_2} = 40 \\ T_{ex}^{M_2} = 4 \end{cases}$$

Analiza offline a setului $\boldsymbol{M}$ presupune un interval de start nul pentru M_1 şi urmăreşte determinarea intervalului de start pentru M_2, în condițiile în care setul este fezabil.

Conform (5-31) şi (5-32), cel mai mare divizor comun, respectiv, cel mai mic multiplu comun al perioadelor FModX-urilor, sunt:

$$\begin{cases} \mathcal{D} = \mathrm{GCD}\left\{T_{pr}^{M_1}, T_{pr}^{M_2}\right\} = \mathrm{GCD}\{30,40\} = 10 \\ \mathcal{M} = \mathrm{LCM}\left\{T_{pr}^{M_1}, T_{pr}^{M_2}\right\} = \mathrm{LCM}\{30,40\} = 120 \end{cases}$$

Primul pas constă din aplicarea condiției de necesitate şi suficiență a planificabilității non-preemptive a setului $\boldsymbol{M}$, conform (5-41):

$$T_{ex}^{M_1} + T_{ex}^{M_2} = 9 < \mathcal{D} = 10.$$

Verificarea condiției implică existența unei planificări fezabile pentru FModX-urile setului $\boldsymbol{M}$.

Aplicarea algoritmului de planificare non-preemptivă a FModX-urilor din setul $\boldsymbol{M}$ constă din calculul valorilor funcției de mapare $\Delta_{M_2/M_1}(\tau)$, pentru $\tau \in \left\{0,1,\ldots,T_{pr}^{M_2} - 1\right\} = \{0,1,\ldots,39\}$ şi contorizarea numărului de instanțe de timp τ pentru care $\Delta_{M_2/M_1}(\tau) = 0$. Se poate observa şi din Figura 5-19 că funcția $\Delta_{M_2/M_1}(\tau)$ este periodică, de perioadă D = 10, şi conține $\nu_{M_2/M_1} = \dfrac{T_{pr}^{M_2}}{\mathcal{D}} = 4$ perioade în care se mapează câte o execuție a lui M_1.

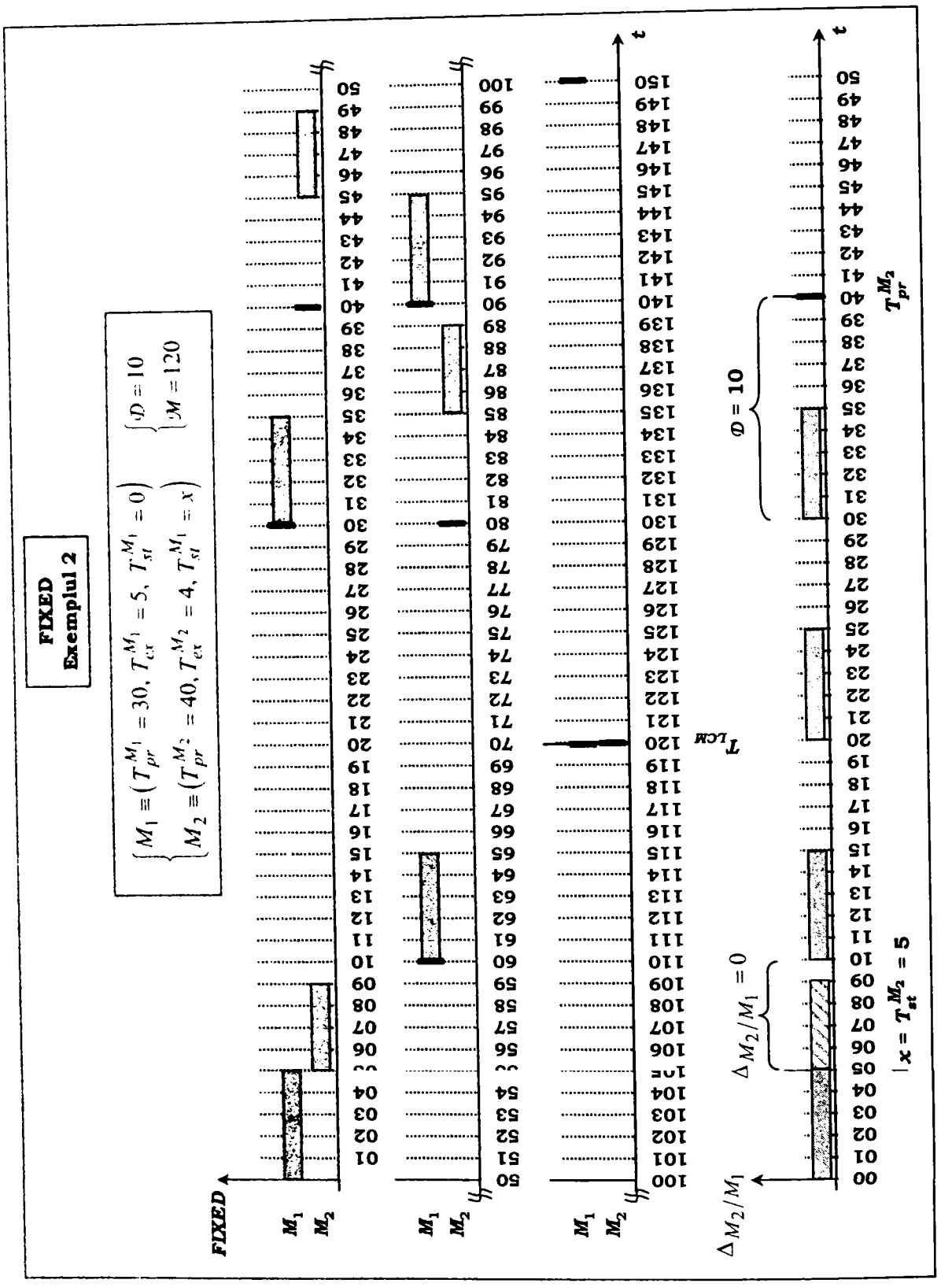

Figura 5-19. Analiza offline a planificării setului ***M***

Cel mai lung interval de timp contiguu pentru care $\Delta_{M_2/M_1}(\tau)=0$ are valoarea T_x = 5, iar prima instanță de timp care defineşte acest interval este t_0 = 5. Mai mult, $T_x = 5 > T_{ex}^{M_2} = 4$. De aici rezultă că, pentru $T_{st}^{M_2} = t_0 = 5$, vom avea o planificare fezabilă a setului ***M*** (vezi Figura 5-19).

Ca observație, menționăm că încărcarea procesor a setului M din acest exemplu are valoarea:

$$U^{\boldsymbol{M}} = \sum_{i=1}^{2} \frac{T_{ex}^{M_i}}{T_{pr}^{M_i}} = 0.266 < 1.$$

❑

5.4.4 Planificarea unui set cu un număr oarecare de FModX-uri

Generalizând principiile de bază prezentate şi demonstrate în paragrafele anterioare, se poate dezvolta analiza planificabilității non-preemptive a unui set ce conține un număr oarecare (mai mare ca 2) de ModX-uri independente, cu execuție fixă.

Teorema 5-6.

Fie ***M*** = $\{M_1, M_2, ..., M_n\}$, un set de *n* FModX-uri independente, ordonate crescător după perioadă:

$$\begin{cases} M_1 \equiv \left(T_{pr}^{M_1}, T_{ex}^{M_1}, T_{st}^{M_1}\right) \\ M_2 \equiv \left(T_{pr}^{M_2}, T_{ex}^{M_2}, T_{st}^{M_2}\right), \text{ cu } T_{pr}^{M_i} \le T_{pr}^{M_j} \text{ pentru } i < j. \\ \vdots \\ M_n \equiv \left(T_{pr}^{M_n}, T_{ex}^{M_n}, T_{st}^{M_n}\right) \end{cases}$$

Setul ***M*** este planificabil în context non-preemptiv, dacă şi numai dacă:

$$\forall\, M_j \in M.\, 1 < j \le n\ ,\ \exists\, t_j \in \left\{0, 1, \ldots, T_{pr}^{M_j} - T_{ex}^{M_j}\right\},\ \text{astfel incat:}$$

$$\bigcup_{\tau=t_0}^{t_j+T_{ex}^{M_j}-1} \bigcup_{i=1}^{j-1} \Delta_{M_j/M_i}(\tau) = 0 \qquad (5\text{-}44)$$

Demonstrație

Teorema afirmă că setul ***M*** este planificabil dacă şi numai dacă orice FModX $M_j \in \boldsymbol{M}$ (j > 1) poate fi planificat împreună cu celelalte FModX-uri anterioare acestuia din punct de vedere al ordonării după perioadă (FModX-urile cu indicii mai mici decât *j*).

Termenul $\bigcup_{i=1}^{j-1}\Delta_{M_j/M_i}(\tau)$ din (5-44) reprezintă maparea în perioada lui M_j a tuturor FModX-urilor din $\boldsymbol{M}$, cu perioade mai mici sau egale cu $T_{pr}^{M_j}$. Prin urmare, teorema susţine că dacă această mapare prevede existenţa unui interval de timp rămas neocupat în care poate avea loc execuţia lui M_j, atunci acest FModX poate fi planificat. Dacă acest lucru se verifică pentru toate M_j din $\boldsymbol{M}$ (cu $j > 1$), atunci setul $\boldsymbol{M}$ este fezabil din punct de vedere al planificării cu execuţie fixă (*FENP*), şi reciproc.

Teorema derivă direct din Teorema 5-5, ca generalizare a acesteia pentru un set $\boldsymbol{M}$ compus din mai mult de două FModX-uri. Demonstraţia este similară cu cea a Teoremei 5-5, considerând cazul $j = n$, adică a FModX-ului cu cea mai mare perioadă din set. Dacă M_n poate fi planificat împreună cu toate celelalte FModX-uri din $\boldsymbol{M}$, rezultă că setul este fezabil, şi reciproc.

❑

Corolarul 5-4.

Dacă setul $\boldsymbol{M} = \{M_1, M_2, ..., M_n\}$ este planificabil conform Teoremei 5-6:

$$\forall M_j \in M.1 < j \le n\ ,\ \exists t_j \in \left\{0,1,\ldots,T_{pr}^{M_j} - T_{ex}^{M_j}\right\},\ \text{astfel incat:}$$

$$\bigcup_{\tau=t_0}^{t_j+T_{ex}^{M_j}-1}\ \bigcup_{i=1}^{j-1}\Delta_{M_j/M_i}(\tau)=0$$

atunci: $T_{st}^{M_j} = t_j$

Demonstraţie

Demonstraţia corolarului derivă direct din demonstraţia Teoremei 5-6.

❑

Corolarul 5-5.

Dacă setul $\boldsymbol{M} = \{M_1, M_2, ..., M_n\}$ de FModX-uri este planificabil, atunci:

$$T_{ex}^{M_i} + T_{ex}^{M_j} \le \mathcal{D}_{i,j} = \mathrm{GCD}\left\{T_{pr}^{M_i}, T_{pr}^{M_j}\right\},\ \forall\, i, j = 1, 2, \ldots, n \qquad (5\text{-}45)$$

Demonstraţie

Demonstraţia corolarului derivă din cea a Corolarului 5-3, care afirmă că dacă există o pereche $\{M_i, M_j\} \subset \boldsymbol{M}$, care nu verifică relaţia (5-45), atunci cele două FModX-uri nu pot fi planificate împreună, rezultând că setul $\boldsymbol{M}$ care le conţine nu este fezabil (conform Teoremei 5-6).

Presupunerea făcută duce însă la contradicţie cu ipoteza corolarului.

❑

Corolarul 5-5 exprimă *condiția de necesitate* a planificării non-preemptive pentru un set de *n* ModX-uri cu execuție fixă (*FENP*):

"*Condiția necesară pentru ca un set de* n *FModX-uri independente să poată fi planificat în context non-preemptiv este ca suma duratelor de execuție a oricăror două FModX-uri din set să nu depășească valoarea celui mai mare divizor comun al perioadelor acestora.*"

Observație

Se poate observa că, spre deosebire de cazul particular al seturilor de două FModX-uri, pentru cazul general ($n > 2$), Corolarul 5-5 furnizează *doar condiția de necesitate*, nu şi de suficiență.

Teorema 5-6, împreună cu Corolarul 5-4 şi 5-5, facilitează abordarea algoritmică a analizei planificabilității non-preemptive *FENP* a unui set $\boldsymbol{M} = \{M_1, M_2, ..., M_n\}$ de FModX-uri ordonate crescător după perioadă. Un astfel de algoritm (care reprezintă practic o generalizare a celui pentru două FModX-uri, ilustrat în Figura 5-18) fixează la început o valoare pentru intervalul de start al lui M_1, $T_{st}^{M_1}$. Dacă nu există alte restricții, $T_{st}^{M_1} = 0$. În continuare se analizează pe rând planificabilitatea celorlaltor FModX-uri din $\boldsymbol{M}$, cu ajutorul relației (5-44). Dacă relația se verifică pentru $M_j \in \boldsymbol{M}$, atunci se stabileşte $T_{st}^{M_j} = t_j$, conform Corolarului 5-4. Algoritmul se termină cu succes dacă toate FModX-urile din $\boldsymbol{M}$ au putut fi planificate.

Pe lângă evaluarea fezabilității setului $\boldsymbol{M}$, un alt rezultat al algoritmului de analiză offline *FENP* constă în determinarea setului $\left\{ T_{st}^{M_i} \middle| M_i \in \boldsymbol{M}, i = 1,2,\ldots,n \right\}$, adică a setului intervalelor de start ale fiecărui FModX din $\boldsymbol{M}$, în cadrul perioadelor corespunzătoare.

Momentul efectiv de start al unui FModX M_i, corespunzător ciclului k de execuție a acestuia, poate fi calculat cu relația:

$$t_{st,k}^{M_i} = (k-1)T_{pr}^{M_i} + T_{st}^{M_i}, \quad k = 1,2,\ldots \tag{5-46}$$

5.5 Concluzii

În capitolul de față am abordat problema planificării non-preemptive din perspectiva modelului de task TR strict – ModX-ul, prezentat în detaliu în capitolul anterior.

A fost arătat faptul că introducerea etapei de analiză offline a planificabilității setului de task-uri, cu confirmarea (sau nu) a fezabilității acestuia, evită problemele de predictibilitate legate de timpii de factură non-polinomială necesari efectuării acestei analize în timpul operării efective a sistemului (analiza online).

Studiul planificării non-preemptive a seturilor de ModX-uri a fost efectuat considerându-se diferite cazuri în care s-au aplicat diverse restricţii, plecând de la situaţiile mai simple (şi mai aproape de ideal) şi avansându-se către situaţii mai complexe şi mai realiste.

Pentru seturile de ModX-uri independente au fost introduşi şi studiaţi doi dintre cei mai eficienţi algoritmi de planificare non-preemptivă: *MLFNP* (Minimum Laxity First Non-Preemptive) şi *EDFNP* (Earliest Deadline First Non-Preemptive). Prin studiile şi simulările efectuate şi exemplificate, am ajuns la concluzia că *EDFNP* poate rezolva cu succes planificarea mai multor seturi de ModX-uri independente, comparativ cu *MLFNP*, fiind astfel mai eficient (aşa cum se va vedea şi în capitolul următor). În acelaşi context, a fost abordată problema evaluării fezabilităţii setului de task-uri utilizând condiţii de necesitate şi/sau suficienţă. Au fost studiate trei condiţii de necesitate a planificabilităţii non-preemptive a seturilor de ModX-uri. Am demonstrat că pentru modelul de task-uri considerat (ModX-uri independente, cu perioadele aliniate la momentul iniţial $t_0 = 0$), condiţia a doua de necesitate enunţată în [Jeffay 91] nu este aplicabilă.

În continuare, am considerat cazurile, mai apropiate de realitate, ale sistemelor TR stricte ce execută, pe lângă setul de ModX-uri ale aplicaţiei, şi un task special – planificatorul online, ce utilizează o Tabelă de Dispatch de dimensiuni fixe, bine determinate, pentru planificarea ModX-urilor. Pentru abordarea planificării online în cicluri cu număr constant de execuţii, au fost concepute şi demonstrate condiţiile de necesitate ale planificabilităţii sistemului, ţinând cont şi de caracteristicile de comportare temporală a planificatorului online. Algoritmul de planificare non-preemptivă pentru astfel de sisteme de task-uri (*CEC-NPOS*) a fost prezentat şi verificat prin simulări şi exemple. Apoi a fost studiată în detaliu abordarea planificării online în cicluri periodice (*PC-NPOS*), demonstrându-se relaţiile ce se aplică parametrilor esenţiali ai planificării (perioada şi durata de execuţie a planificatorului online, şi dimensiunea maximă a Tabelei de Dispatch).

În fine, am introdus şi abordat problema modelării şi planificării seturilor de task-uri ce necesită o execuţie fixă în cadrul fiecărei perioade a acestora. A fost conceput şi studiat modelul matematic al ModX-urilor cu execuţie fixă (FModX-uri). A fost analizată planificarea non-preemptivă a seturilor compuse din două FModX-uri independente, prin introducerea şi demonstrarea condiţiilor de necesitate şi suficienţă ale planificabilităţii, şi prin conceperea algoritmului de planificare ce utilizează funcţia de mapare (*FENP*). Modul de operare al algoritmului a fost exemplificat pe diverse cazuri. Problema a fost apoi generalizată pentru un set cu un număr oarecare (mai mare ca 2) de FModX-uri.

Preocupările curente şi de viitor imediat din cadrul activităţii noastre de cercetare în domeniu cuprind următoarele subiecte legate de planificarea non-preemptivă a seturilor de ModX-uri:

- Planificarea seturilor de ModX-uri cu dependenţe de program (cu alte cuvinte, planificarea unei aplicaţii TR complete);
- Integrarea tuturor cazurilor de planificare non-preemptivă şi conceperea unui model unificat, capabil a fi utilizat în cât mai multe aplicaţii reale.

6 Evaluarea performanţelor algoritmilor de planificare non-preemptivă

În cadrul capitolului anterior au fost discutaţi în detaliu o serie de algoritmi de planificare non-preemptivă existenţi în literatura de specialitate, cu particularizările impuse de utilizarea modelului de task TR strict introdus de abordarea noastră: ModX-ul. De asemenea, în Capitolul 5 au fost introduşi şi studiaţi noi algoritmi de planificare non-preemptivă, pentru rezolvarea diverselor cazuri reale, ce se abat de la modelul abstract, general, utilizat în publicaţiile ce tratează sistemele TR. De exemplu, pentru tratarea planificării online, au fost introduşi algoritmii *CEC-NPOS* şi *PC-NPOS*, iar pentru planificarea seturilor de ModX-uri cu execuţie fixă în cadrul perioadei (FModX-uri) a fost definit algoritmul *FENP*.

Datorită importanţei planificării în cadrul STR stricte şi a impactului pe care aceştia îl au asupra performanţei întregului sistem, o analiză amănunţită a performanţelor acestor algoritmi este obligatorie [Lehoczky 89, Panzieri 93a, Panzieri 93b, Ghosh 94]. Pentru cazurile ce pot fi tratate de mai mulţi algoritmi de planificare, studiul comparativ al acestora permite luarea unor decizii de selectare a celui mai performant.

În paragrafele următoare sunt prezentate evaluările de performanţă ale următorilor algoritmi de planificare non-preemptivă:

1. *EDFNP* şi *MLFNP*: studiu comparativ pentru evaluarea planificării non-preemptive a seturilor de ModX-uri simple, independente.
2. *CEC-NPOS* şi *PC-NPOS*: studiu comparativ pentru evaluarea planificării online non-preemptive a seturilor de ModX-uri simple, independente, utilizând o Tabelă de Dispatch de dimensiune maximă prestabilită.
3. *FENP*: analiza performanţelor planificării non-preemptive a seturilor de FModX-uri independente.

6.1 Introducere: metodele utilizate pentru analiza performanţelor

Evaluarea performanţelor algoritmilor de planificare propuşi se bazează pe determinarea următorilor parametri generali:

- Succesul sau insuccesul condiţiilor de planificabilitate care se aplică algoritmului studiat;
- Succesul sau insuccesul planificării efectuate cu algoritmul studiat asupra unui set dat de ModX-uri. Fezabilitatea planificării se analizează aplicând algoritmul respectiv asupra setului de ModX-uri pentru intervalul temporal $[0, T_{LCM})$, unde T_{LCM} reprezintă cel mai mic multiplu comun al perioadelor ModX-urilor din setul analizat (conform Lemei 5-1);

- Timpul efectiv necesar execuţiei algoritmului pentru terminarea întregii planificări a unui set de ModX-uri, pe un calculator PC tip Pentium III. Acest timp nu este un parametru esenţial în ceea ce priveşte performanţele algoritmilor de planificare studiaţi, pentru că aici este tratată analiza de tip offline a seturilor de ModX-uri aparţinând sistemului şi aplicaţiilor. Comportarea algoritmilor în timpul operării sistemului (la "run-time") vizează corectitudinea operării propriu-zise a algoritmului de planificare, durata sa de execuţie pentru un singur ciclu de planificare, şi (dacă e cazul) perioada acestuia. Aceşti parametri se determină după implementarea planificatorului online pe platforma ţintă, după ce a fost selectat algoritmul potrivit (sau combinaţia de algoritmi).

Pentru analiza performanţelor algoritmilor de planificare studiaţi, se efectuează sesiuni extinse de teste asupra unor seturi de ModX-uri generate aleator în ceea ce priveşte parametrii temporali ai ModX-urilor. Astfel, programele de evaluare vor primi o configuraţie de operare, ce specifică printre altele:

- `TotMX` (= n): numărul total de ModX-uri ce compun seturile analizate;
- `IntervTimp`: intervalul temporal în care se vor genera perioadele ModX-urilor din set;
- `GenPer`: tipul de distribuţie utilizată la generarea aleatoare a valorilor perioadelor ModX-urilor;
- [`MinPU, MaxPU`]: intervalul de valori ce încadrează factorul de utilizare procesor al setului $\boldsymbol{M}$ de ModX-uri generate, $PU = U^M$. Acest parametru influenţează duratele de execuţie ale ModX-urilor din $\boldsymbol{M}$;
- `TschedMax` = T_{LCM} : intervalul temporal maxim în care se poate încadra limita de planificare;
- `NrTeste`: numărul total de teste de planificare (numărul total de seturi generate aleator pentru analiza fezabilităţii cu un anumit algoritm).

Exemplul 6-1.

Considerăm pentru programul de evaluare, configuraţia dată în Tabelul 6-1.

Tabelul 6-1. Exemplu de configuraţie pentru programul de evaluare

Parametru configuraţie	Valoare
`TotMX`	10
`IntervTimp`	310
`GenPer`	2
`[MinPU, MaxPU]`	[0.8, 0.9]
`TschedMax`	100000000
`NrTeste`	1000

Ca urmare, evaluarea algoritmului de planificare studiat se va face cu 1000 de teste individuale, în care se generează aleator câte un set de 10 ModX-uri. Parametrii temporali ai fiecărui ModX din cele 1000 de seturi ce vor fi generate,

adică perioada şi durata de execuție, vor avea următoarele caracteristici: perioada se va încadra în intervalul [10, 310], fiind obținută prin aplicarea unei proceduri de generare aleatoare cu distribuție normală (Gaussiană) pentru acest interval, iar durata de execuție va respecta condiția ca utilizarea procesor a întregului set să fie cuprinsă în intervalul [0.8, 0.9]. De asemenea, cel mai mic multiplu comun al perioadelor ModX-urilor din fiecare set va fi mai mic decât 100000000.

Duratele de execuție a ModX-urilor sunt obținute de asemenea în manieră aleatoare, verificându-se de fiecare dată utilizarea procesor a întregului set:

$$U^M = \sum_{i=1}^{n} \frac{T_{ex}^{M_i}}{T_{pr}^{M_i}}$$

Tabelul 6-2 prezintă un exemplu de set de ModX-uri generat pe baza parametrilor de configurare din Tabelul 6-1.

Tabelul 6-2. Exemplu de set de ModX-uri generat

Element	**Valoare**	
	Perioada	**Durata execuție**
M_1	63	1
M_2	95	12
M_3	126	15
M_4	133	16
M_5	133	9
M_6	141	17
M_7	172	10
M_8	210	18
M_9	215	22
M_{10}	235	1
U^M	0.820208	
T_{LCM}	48382740	

❑

Pentru generarea aleatoare a perioadelor ModX-urilor au fost utilizate două tipuri distincte de distribuție: (a) distribuția uniformă şi (b) distribuția normală (Gaussiană). Procedura de generare uniform aleatoare a perioadelor ModX-urilor împarte intervalul fixat pentru acestea la configurare, [10, `IntervTimp`], în *n* (`TotMX`) subintervale de dimensiune egală, din care este apoi extrasă câte o valoare. Figura 6-1 prezintă histogramele şi liniile de tendință a evoluției ("trend lines") pentru patru seturi de câte 50 de ModX-uri, cu perioade generate uniform aleator în intervalul [10, 310].

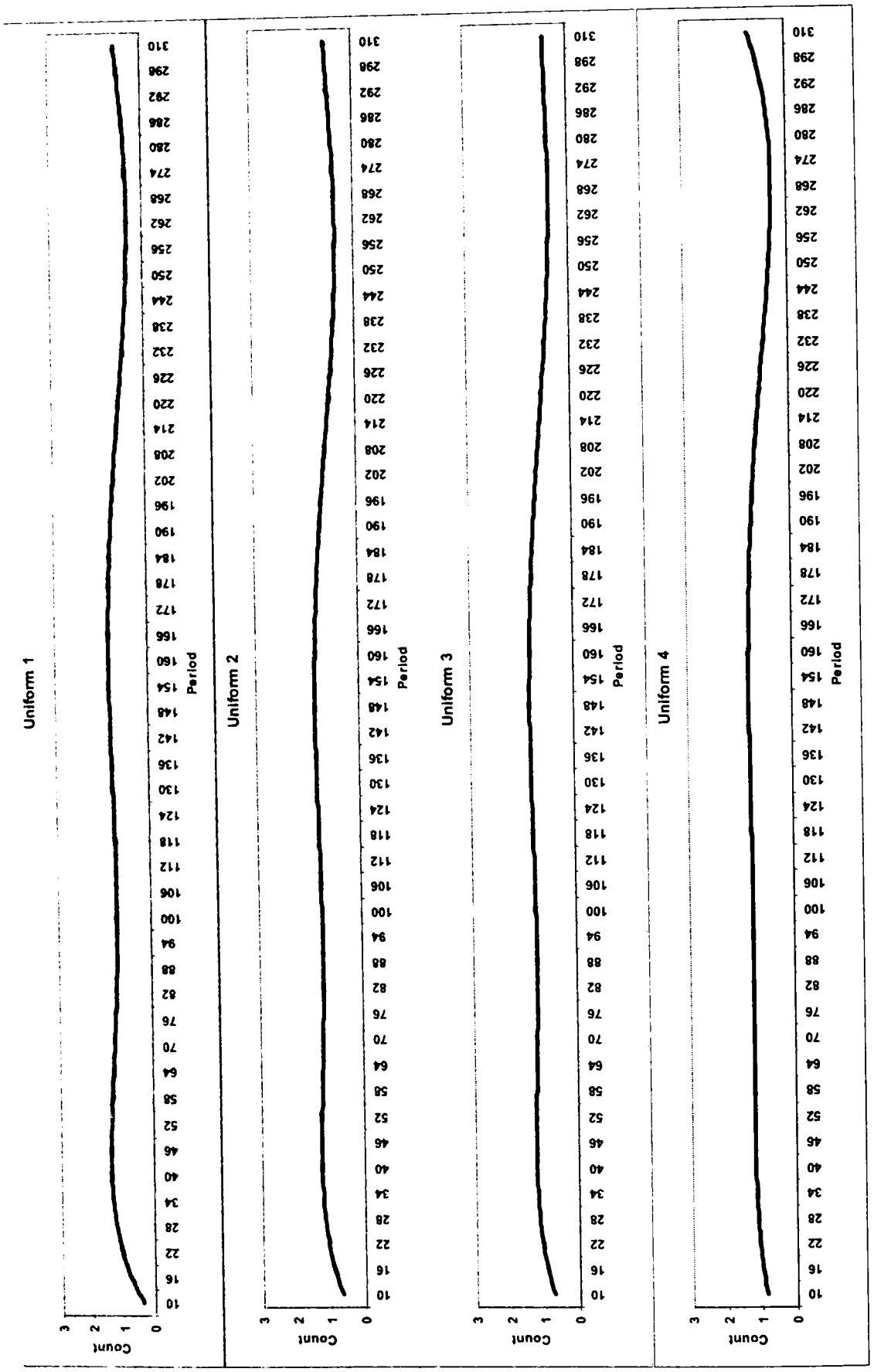

Figura 6-1. Histogramele a patru seturi de ModX-uri generate cu distribuţie uniformă

Generarea de numere aleatoare cu distribuţie normală (Gaussiană) se bazează pe funcţia de densitate normală [Knuth 98, Ulrich 98]:

$$N(x,\mu,\sigma)=\frac{1}{\sigma\sqrt{2\pi}}e^{\frac{-(x-\mu)^2}{2\sigma^2}} \tag{6-1}$$

unde μ reprezintă media de distribuţie iar σ este deviaţia standard. Funcţia de densitate normală standard se obţine pentru o medie nulă şi o deviaţie standard unitară, şi reprezintă probabilitatea de generare a valorilor numerice din intervalul [–3, 3] (vezi Figura 6-2):

$$N(x,0,1)=\frac{1}{\sqrt{2\pi}}e^{\frac{-x^2}{2}} \tag{6-2}$$

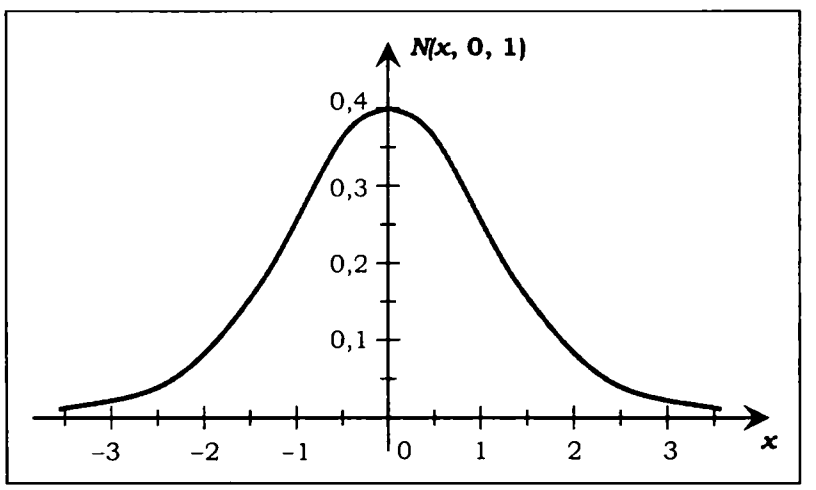

Figura 6-2. Graficul distribuţiei normale standard, $N(x, 0, 1)$

Generarea numerică a valorilor cu distribuţie normală standard utilizează forma polară a transformării Box-Muller [Carter 04], pe baza următorului algoritm:

Programul 6-1. Procedura de generare a valorilor cu distribuţie normală standard

```
float x1, x2, w, y1, y2;
do {
   x1 = 2.0 * rand() - 1.0;
   x2 = 2.0 * rand() - 1.0;
   w = x1 * x1 + x2 * x2;
   } while (w >= 1.0);
w = sqrt((-2.0 * ln(w))/w);
y1 = x1 * w;
y2 = x2 * w;
```

Cu algoritmul de mai sus se obţine o pereche de valori {`y1`, `y2`} normal distribuite în intervalul [–1, 1]. Funcţia `rand()` se găseşte în bibliotecile standard ANSI C, pentru generarea de numere aleatoare uniform distribuite în intervalul [–1, 1].

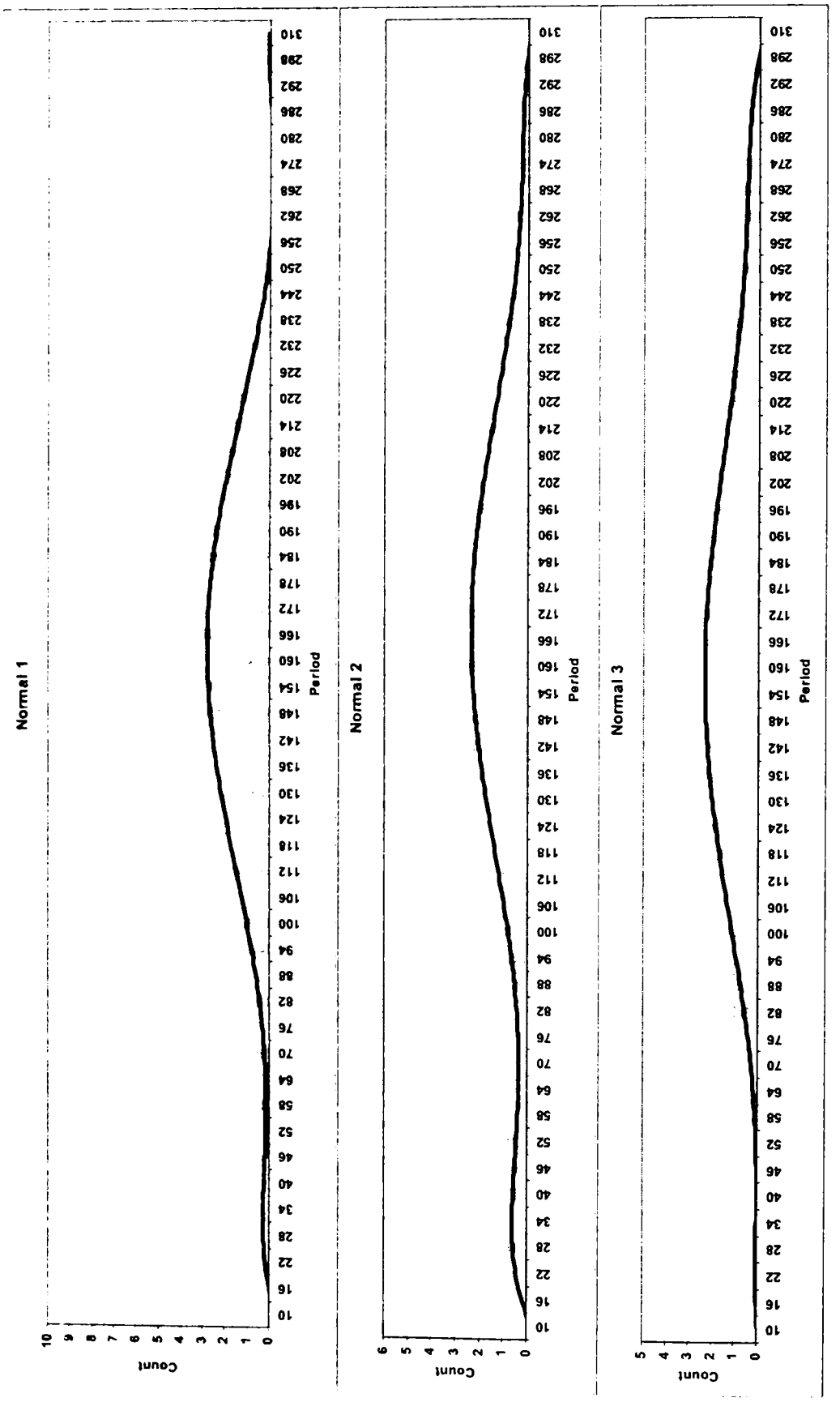

Figura 6-3. Histogramele a trei seturi de ModX-uri generate cu distribuţie normală

În final, pentru generarea perioadelor ModX-urilor în intervalul specificat la configurarea programului, [10, `IntervTimp`], se va efectua transformarea de la (6-2) la (6-1):

$$T_{pr}^{M_i} = \mu + \sigma \cdot \mathtt{y1}, \text{ unde: } \begin{cases} \mu = \dfrac{\mathtt{IntervTimp} - 10}{2} + 10 \\ \sigma = \dfrac{\mathtt{IntervTimp} - 10}{6} \end{cases} \quad (6\text{-}3)$$

Figura 6-3 prezintă histogramele şi liniile de tendinţă a evoluţiei ("trend lines") pentru trei seturi de câte 50 de ModX-uri, cu perioade generate uniform aleator în intervalul [10, 310]. Parametrii procedurii de determinare a perioadelor corespunzători configuraţiei din Figura 6-3 sunt: μ = 160 şi σ = 50.

Evaluarea performanţelor algoritmilor de planificare pentru modelul de STR propus şi discutat în lucrarea de faţă presupune abordarea a două perspective: (i) selectarea unui algoritm cu performanţe optime în ceea ce priveşte planificarea a cât mai multe seturi de ModX-uri posibil şi pentru încărcări procesor cât mai mari, şi (ii) evaluarea performanţelor de operare online ale aceluiaşi algoritm. În cel de al doilea caz, parametrii vizaţi sunt în legătură directă cu modul de implementare a codului planificatorului online: durata maximă de execuţie (viteza) şi dimensiunea codului. În Secţiunea III sunt discutate în detaliu aceste aspecte.

Un parametru cheie de performanţă în ceea ce priveşte analiza optimalităţii algoritmilor de planificare, este aşa-numitul *raport de planificabilitate* (*SR*, *Schedulability Ratio*). Raportul de planificabilitate va fi evaluat în paragrafele următoare pentru diferiţi algoritmi de planificare şi pentru diferite încărcări procesor implicate de seturi de ModX-uri generate aleator, cu tipurile de distribuţie prezentate anterior.

6.2 Evaluarea comparativă a algoritmilor EDFNP şi MLFNP

Pentru evaluarea comparativă a performanţelor (optimalităţii) algoritmilor de planificare non-preemptivă *EDFNP* şi *MLFNP* discutaţi în Capitolul 5, a fost dezvoltat un program specific de testare, "Evaluation1" şi un program de colectare a rezultatelor, "DataAdd", care are şi rolul de eliminare a rezultatelor duplicat.

Cu ajutorul programului "Evaluation1" au fost efectuate peste 24000 de teste, utilizând cele 12 staţii de calcul ale reţelei de calculatoare tip PC Pentium III a laboratorului DSPLabs (http://dsplabs.utt.ro). Testele desfăşurate au urmărit determinarea raportului de planificabilitate *SR* în funcţie de următorii parametri suplimentari (combinaţii ale acestora):

- Numărul total de ModX-uri dintr-un set: `TotMX` = 10, 15, 20;
- Utilizarea procesor (*PU*) implicată de seturile de ModX-uri, încadrată în intervalul: [`MinPU`, `MaxPU`] = [0.6, 0.7], [0.7, 0.8], [0.8, 0.9], [0.9, 1.0];
- Tipul de distribuţie utilizat la generarea aleatoare a perioadelor ModX-urilor: distribuţie uniformă, şi distribuţie normală.

Intervalul ce încadrează valorile perioadelor ModX-urilor a fost considerat constant: `IntervTimp` = 310, valoarea minimă pentru perioade fiind 10. Cum valoarea minimă a duratelor de execuţie a ModX-urilor generate este 1, rezultă un raport maxim de 1/310 între durata de execuţie şi perioada oricărui ModX. Seturile de ModX-uri utilizate în operaţiile de evaluare pot fi puse în corespondenţă cu cazurile realiste din aplicaţiile TR, prin aplicarea unor factori de scalare pentru parametrii temporali respectivi. Astfel, de exemplu, un ModX $M_i \equiv \left\{ T_{pr}^{M_i} = 210,\ T_{ex}^{M_i} = 3 \right\}$ ce este utilizat în programul de evaluare, poate fi pus în relaţie directă cu ModX-ul $M_i' \equiv \left\{ 100\,T_{pr}^{M_i} = 21000\ \mu s,\ 100\,T_{ex}^{M_i} = 300\ \mu s \right\}$ al unei aplicaţii TR reale, prin scalarea cu 100 μs a parametrilor temporali ai lui M_i.

Pe lângă evaluarea celor doi algoritmi studiaţi, *EDFNP* şi *MLFNP*, programul "Evaluation1" aplică şi testarea condiţiei de necesitate a planificării **CN2)** (vezi Capitolul 5), introdusă de [Jeffay 91]. Aşa cum s-a demonstrat în capitolul anterior, această condiţie de necesitate nu este valabilă pentru toate seturile de ModX-uri, deoarece modelul de seturi de task-uri TR utilizat aici este un caz particular faţă de modelul considerat în lucrarea lui Jeffay.

În Figura 6-4, Figura 6-5 şi Figura 6-6 sunt prezentate graficele rezultate ca urmare a testelor de planificabilitate aplicate cu programul "Evaluation1" şi procesate apoi cu "DataAdd". Din cele trei figuri se pot desprinde următoarele elemente:

- *EDFNP* se comportă mai bine ca *MLFNP* în sensul unei rate de succes a planificării mai mari, pentru toate cazurile considerate: orice total de ModX-uri din set, orice interval de încărcare procesor şi orice tip de distribuţie utilizată la generarea parametrilor temporali ai ModX-urilor;
- *EDFNP* reuşeşte planificarea multor seturi de ModX-uri pentru care condiţia **CN2)** (testul "Jeffay") dă rezultate negative. Acest lucru se observă mai pregnant pentru seturile cu un număr mai mic de ModX-uri;
- Pentru oricare din cazurile considerate, seturile de ModX-uri care dau rezultate pozitive la testul "Jeffay", pot fi planificate cu algoritmul EDFNP. Există însă şi seturi particulare ce pot fi planificate cu EDFNP şi fără a îndeplini condiţia "Jeffay". Acest lucru este în concordanţă cu discuţia din Capitolul 5 referitoare la condiţia de planificabilitate **CN2)**, unde a fost evidenţiată şi demonstrată teoretic această problemă;
- Comportarea ambilor algoritmi studiaţi se îmbunătăţeşte vizibil pe măsură ce numărul de ModX-uri din seturile supuse planificării, creşte. Acest lucru se explică prin faptul că, pentru o aceeaşi încărcare procesor (*PU*), cu cât sunt mai multe ModX-uri într-un set, cu atât scade durata de execuţie a acestora. Prin urmare planificarea non-preemptivă are şanse mai mari de reuşită pentru mai multe task-uri, ce se execută fiecare într-un interval mai scurt, decât dacă sunt task-uri mai puţine dar de durată mai mare;
- Rata de succes a planificării cu cei doi algoritmi scade, aşa cum este de aşteptat, pe măsură ce creşte utilizarea procesor a seturilor de ModX-uri testate;

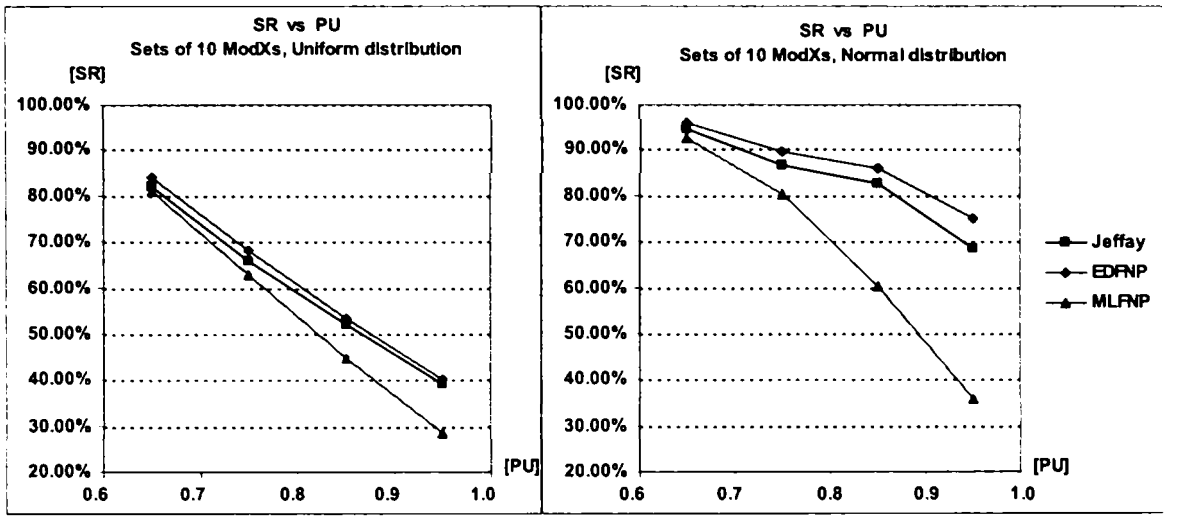

Figura 6-4. *SR* funcție de *PU*, pentru seturi de 10 ModX-uri cu distribuție uniformă (stânga) şi normală (dreapta)

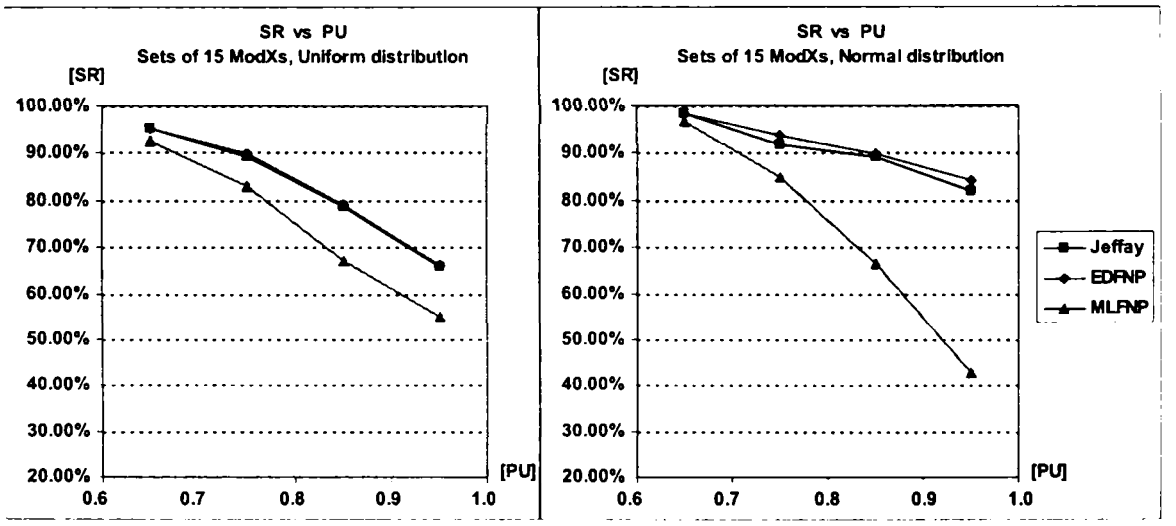

Figura 6-5. *SR* funcție de *PU*, pentru seturi de 15 ModX-uri

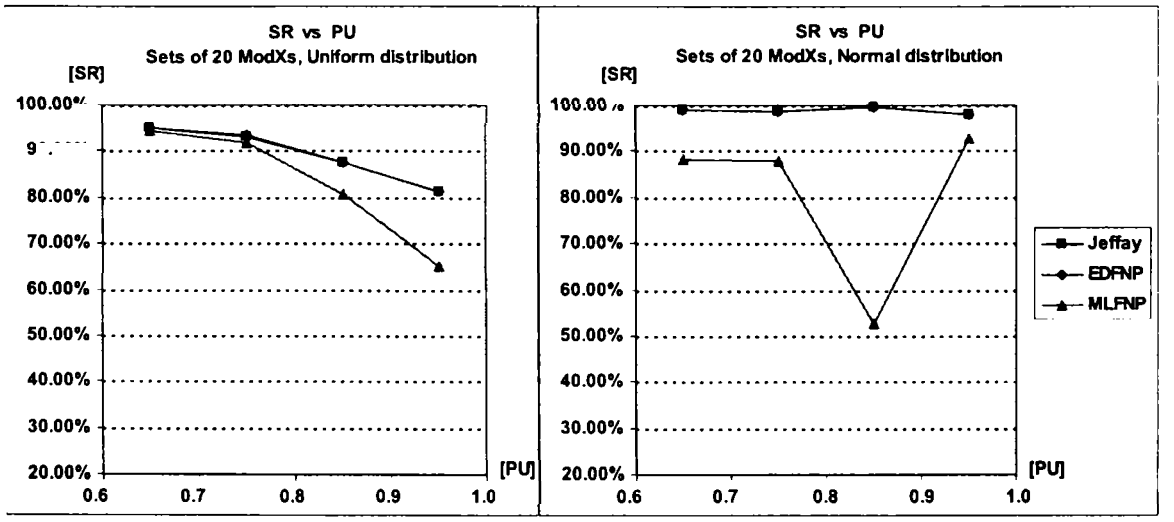

Figura 6-6. *SR* funcție de *PU*, pentru seturi de 20 ModX-uri

- În cazul seturilor de ModX-uri generate cu distribuţie normală, ambii algoritmi se comportă semnificativ mai bine, comparativ cu seturile generate cu distribuţie uniformă. Acest lucru se explică prin faptul că algoritmii de planificare non-preemptivă discutaţi dau rezultate mai bune pentru seturi de ModX-uri cu perioade având valori relativ apropiate între ele, situate în apropierea mediei intervalului considerat pentru acestea, comparativ cu cazurile în care perioadele iau valori extreme, mult diferite între ele. Distribuţia normală a perioadelor în cadrul unui anumit interval corespunde de asemenea multor aplicaţii TR practice.

În Figura 6-6 se poate observa, cu precădere pentru distribuţia normală, o deviere de la tendinţa generală de comportare a algoritmilor *EDFNP* şi *MLFNP* (Figura 6-4 şi Figura 6-5). Explicaţia constă în faptul că în procesul de evaluare a performanţelor cu ajutorul programului "Evaluation 1", a fost impusă o limită pentru valoarea timpului de planificare: T_{LCM} = `TschedMax` = 1000000000 (pentru seturile de 20 ModX-uri). Ca rezultat, găsirea (în manieră aleatoare) a unui set de 20 de valori pentru perioadele ModX-urilor, astfel încât cel mai mic multiplu comun al acestora să se încadreze în limita impusă, este dificilă şi de durată. Cu cât valorile perioadelor sunt mai mari (ca în cazul distribuţiei normale, Figura 6-6 dreapta, faţă de distribuţia uniformă, Figura 6-6, stânga), cu atât mai dificilă este această problemă.

Limitarea timpului maxim de planificare (T_{LCM}) a fost impusă deoarece în caz contrar, se pot genera cu uşurinţă seturi de 20 de ModX-uri de exemplu, pentru care T_{LCM} să ajungă la ordinul de mărime 10^{30}, şi chiar mai mult. De aici rezultă două probleme:

(a) Necesitatea operării cu numere întregi foarte mari, care depăşesc capacităţile tipurilor de date implementate pe majoritatea arhitecturilor PC de către limbajele de programare. În consecinţă se impune utilizarea de biblioteci specializate în operarea cu numere cu precizie variabilă [Shoup 02, Allombert 03, Shamus 03].

(b) Timpul necesar analizei planificabilităţii seturilor de ModX-uri este direct proporţional cu T_{LCM}. Tabelul 6-3 prezintă timpii de planificare rezultaţi ca urmare a aplicării programului "Evaluation 1" pentru câteva seturi de ModX-uri. Programul a fost rulat pe o staţie PC tip Pentium III, pentru cinci seturi de 18 ModX-uri, cu perioadele generate cu distribuţie normală în intervalul [10, 310] astfel încât limita $T_{LCM} \leq 2000000000$. Utilizarea procesor a seturilor a fost aleasă cât mai mică (în intervalul [0.1, 0.2]) pentru ca timpii rezultaţi să fie maximi pentru perioadele respective ale ModX-urilor.

Cele două probleme de mai sus converg până la urmă tot la dificultăţi legate de timpul de procesare. Utilizarea unei biblioteci pentru numere cu precizie variabilă implică o creştere a acestor timpi cu cel puţin un ordin de mărime.

Versiunea utilizată în final pentru programul "Evaluation 1" la efectuarea testelor comparative prezentate în diagramele de mai sus, se bazează pe o implementare Microsoft® VisualStudio™ 6.0 (Visual C++), utilizând tipul de numere întregi reprezentate pe 64 biţi: `__int 64`. În acest fel se pot efectua operaţii rapide cu numere întregi de ordinul $9 \cdot 10^{18}$.

Tabelul 6-3. Timpii de analiză a planificării pentru câteva teste cu "Evaluation1"

Valoare T_{LCM}	Timpi de planificare [sec]	
	MLFNP	*EDFNP*
145044900	476	469
325155600	1060	1052
149189040	483	481
681912000	2214	2212
1730907360	5698	5601
Medieri		
1000000000	3275	3237

6.3 Evaluarea comparativă a algoritmilor de planificare online non-preemptivă

Comportarea algoritmilor de planificare online non-preemptivă introduşi şi discutaţi în Subcapitolul 5.3, *CEC-NPOS* şi *PC-NPOS*, a fost testată şi analizată cu ajutorul unui program special conceput şi dezvoltat cu mediul Microsoft® VisualStudio™ 6.0 (Visual C++) pentru platforme PC: "OnlineScd". Principiile de evaluare implementate de "OnlineScd" sunt similare cu cele ale lui "Evaluation 1": efectuarea unui număr prestabilit de teste de planificare pentru algoritmii *CEC-NPOS* şi *PC-NPOS*, asupra unor seturi cu număr dat de ModX-uri, ai căror parametri temporali sunt generaţi în manieră aleatoare după distribuţii uniforme sau normale.

Elementele suplimentare ce apar în acest caz sunt ModX-ul sistem de planificare online, M_S, şi Tabela de Dispatch a sistemului TR, modelată prin dimensiunea maximă a înregistrărilor sale, $\lambda + 1$.

Procedura generală de operare a programului "OnlineScd" cuprinde următorii paşi:

1. Se generează perioadele ModX-urilor normale $M_i \in \boldsymbol{M} = \{M_1, M_2, \ldots, M_n\}$, unde: $n =$ `TotMX` este totalul ModX-urilor din $\boldsymbol{M}$, perioadele sunt cuprinse în intervalul [10, `IntervTimp`] şi sunt generate aleator, cu distribuţie uniformă sau normală, astfel încât cel mai mic multiplu comun al lor T_{LCM} să nu depăşească valoarea `TschedMax`.

2. Se sortează ModX-urile din setul $\boldsymbol{M}$ în ordinea crescătoare a perioadelor.

3. Se generează duratele de execuţie ale ModX-urilor din $\boldsymbol{M}$, astfel încât utilizarea procesor a setului să fie cuprinsă în intervalul: $PU_M \in$ {`MinPU`, `MaxPU`}.

4. Se generează durata de execuţie a planificatorului online M_S în funcţie de perioada şi durata de execuţie a ModX-ului cu cea mai mică perioadă din set:

$$T_{ex}^{M_S} = 2\left(T_{pr}^{M_{pr\min}} - T_{ex}^{M_{pr\min}}\right) \tag{6-4}$$

La valoarea obţinută pentru durata de execuţie a M_S, se aplică un factor de scalare (subunitar), specificat la configurarea programului: `Scale_TexMs`.

5. În continuare, în funcţie de tipul algoritmului de planificare aplicat, se parcurg următoarele etape:
   5.1. Pentru *CEC-NPOS*:
   - se verifică şi se contorizează condiţia de planificabilitate *CEC-NPOS* (Teorema 5-2);
   - se aplică şi se contorizează algoritmul de planificare, indiferent de rezultatele condiţiei de necesitate.

   5.2. Pentru *PC-NPOS*:
   - se determină perioada ModX-ului de planificare online M_S, cu relaţia (5-26) şi procedura descrisă în Subcaptiolul 5.3.2;
   - se re-sortează ModX-urile din set, împreună cu M_S, după perioade;
   - se aplică şi se contorizează testul "Jeffay" pentru setul extins de ModX-uri: $\boldsymbol{M} \cup \{M_S\}$;
   - se aplică şi se contorizează algoritmul de planificare *PC-NPOS*, indiferent de rezultatul testului "Jeffay".

Cu ajutorul programului "OnlineScd" au fost efectuate peste 35000 de teste, utilizând cele 12 staţii ale reţelei de calculatoare tip PC Pentium III a laboratorului DSPLabs. Testele desfăşurate au urmărit determinarea raportului de planificabilitate *SR* în funcţie de următorii parametri suplimentari (combinaţii ale acestora):

- Numărul total de ModX-uri dintr-un set: `TotMX` = 10, 15;
- Utilizarea procesor (*PU*) implicată de seturile de ModX-uri, încadrată în intervalul: [`MinPU`, `MaxPU`] = [0.5, 0.6], [0.6, 0.7], [0.7, 0.8], [0.8, 0.9], şi [0.9, 1.0];
- Dimensiunea maximă utilă a Tabelei de Dispatch: `lambda` = 100, 200;
- Factorul de scalare a duratei de execuţie a M_S: `Scale_TexMs` = 0.3, 0.5, 0.7 .

Intervalul ce încadrează valorile perioadelor ModX-urilor a fost considerat constant: `IntervTimp` = 310, valoarea minimă pentru perioade fiind 10. În Figura 6-7 la Figura 6-12 sunt prezentate diagramele rezultatelor testelor efectuate cu "OnlineScd". Din aceste diagrame rezultă ca evidentă comportarea mult mai bună a algoritmului de planificare online periodică non-preemptivă, *PC-NPOS*, pentru toate cazurile considerate. Pe de altă parte, atunci când durata de execuţie a planificatorului M_S se apropie de valoarea maximă admisă de condiţia de necesitate **CN3)** (relaţia (6-4)), de exemplu pentru un factor de scalare de 0.7 (Figura 6-9), rata de succes a algoritmilor online este foarte mică.

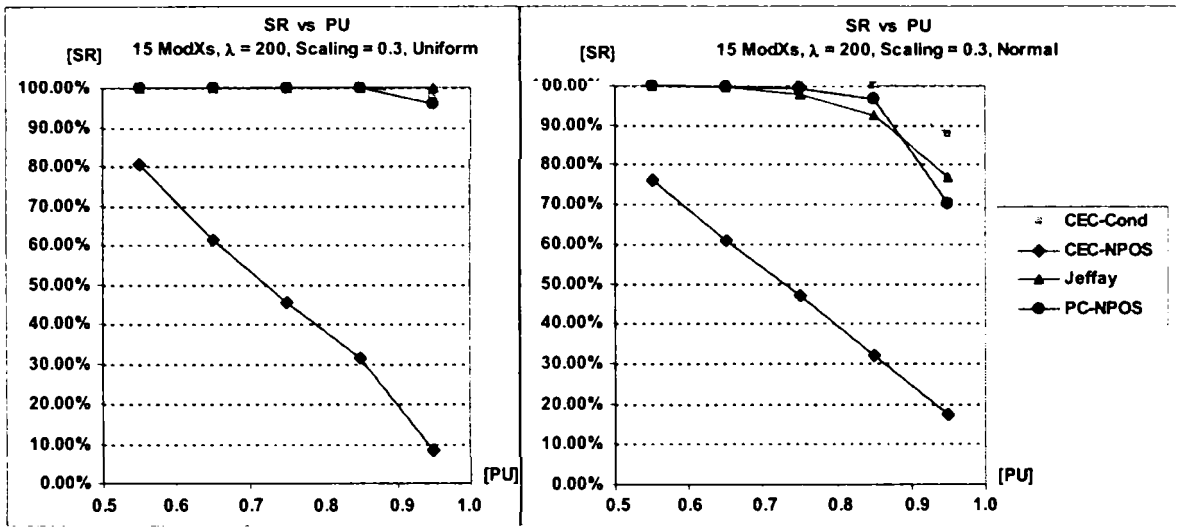

Figura 6-7. *SR* funcție de *PU*, pentru 15 ModX-uri, $\lambda = 200$ şi factorul de scalare 0.3, distribuție uniformă (stânga) şi normală (dreapta)

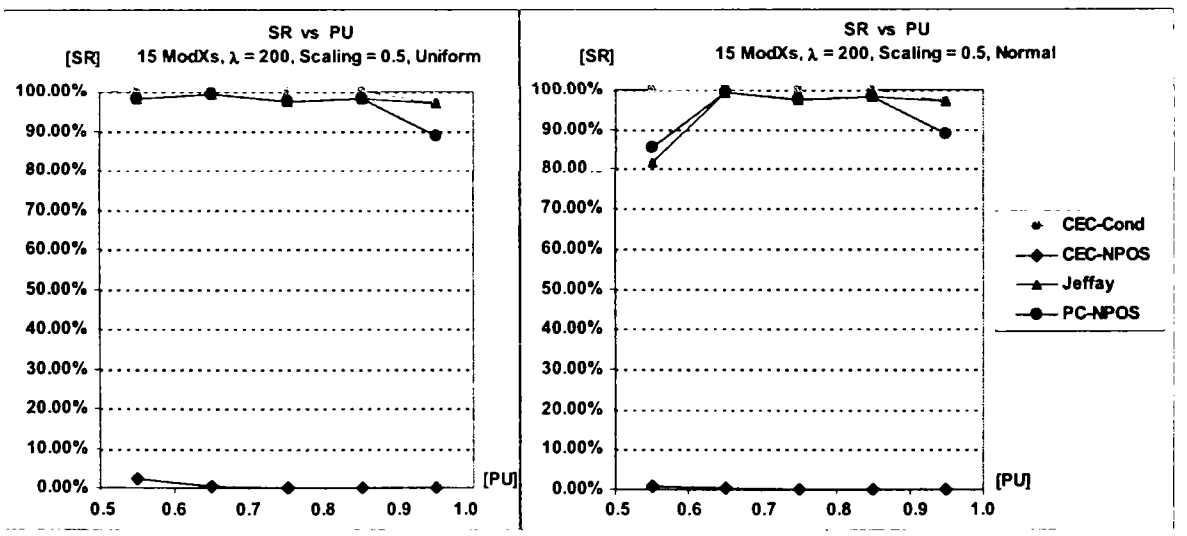

Figura 6-8. *SR* funcție de *PU*, pentru 15 ModX-uri, $\lambda = 200$ şi factorul de scalare 0.5

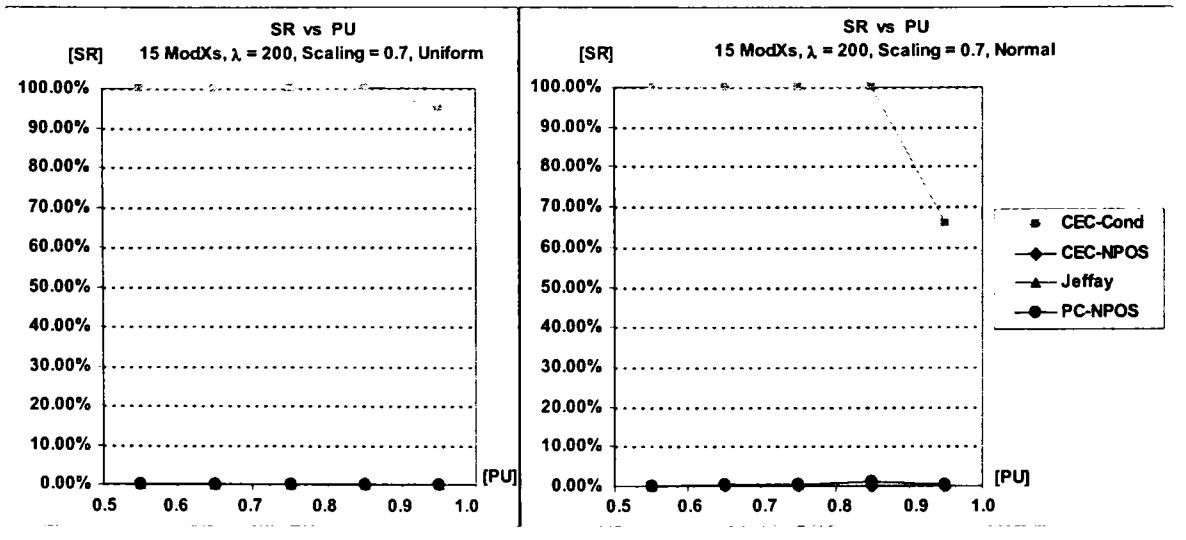

Figura 6-9. *SR* funcție de *PU*, pentru 15 ModX-uri, $\lambda = 200$ şi factorul de scalare 0.7

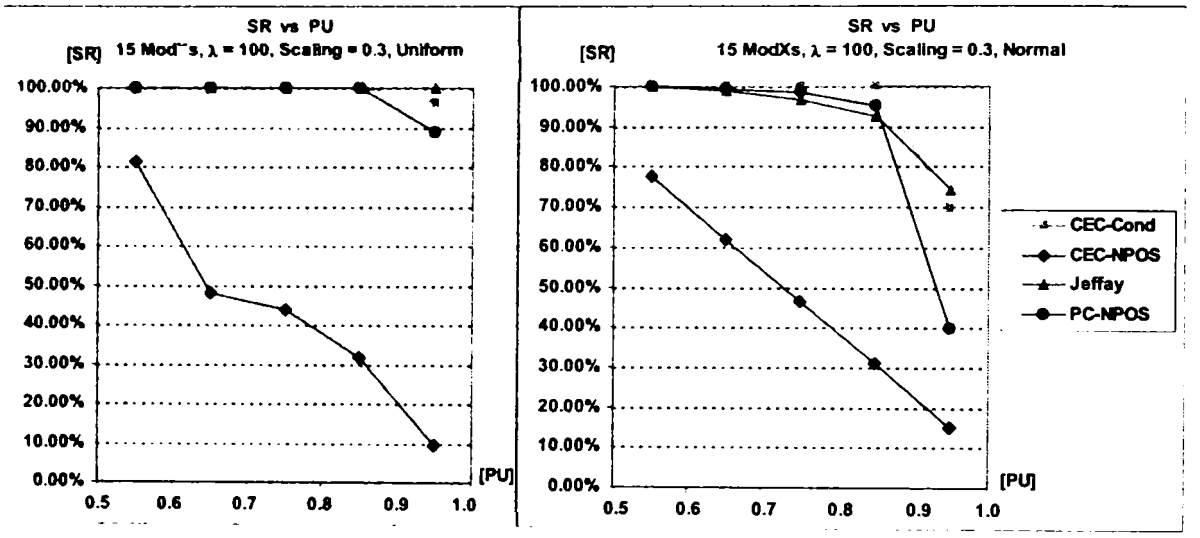

Figura 6-10. *SR* funcţie de *PU*, pentru 15 ModX-uri, $\lambda = 100$ şi factor de scalare 0.3

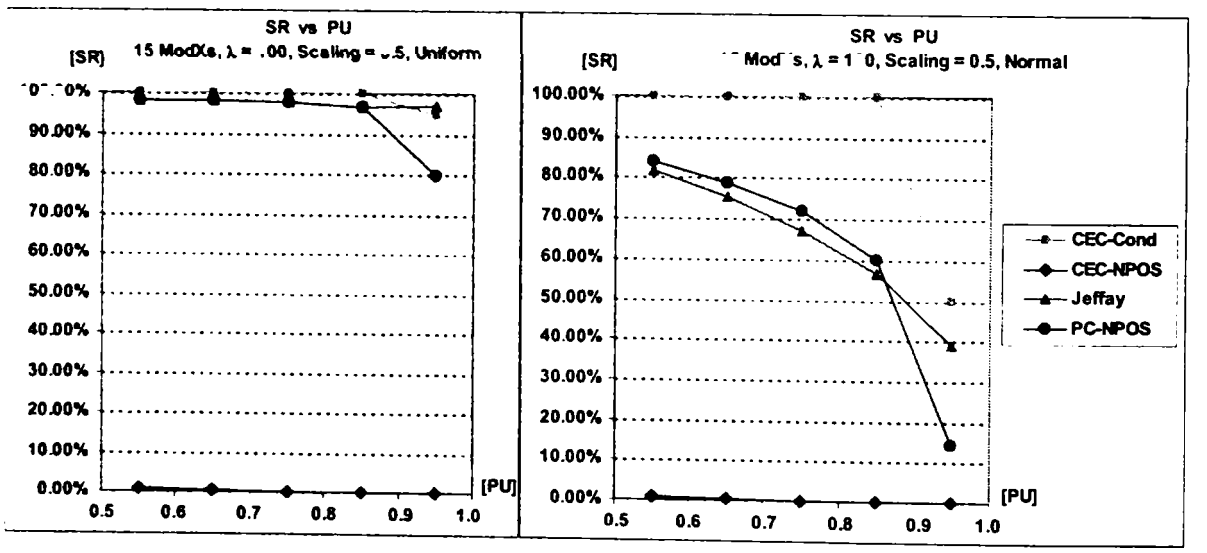

Figura 6-11. *SR* funcţie de *PU*, pentru 15 ModX-uri, $\lambda = 100$ şi factor de scalare 0.5

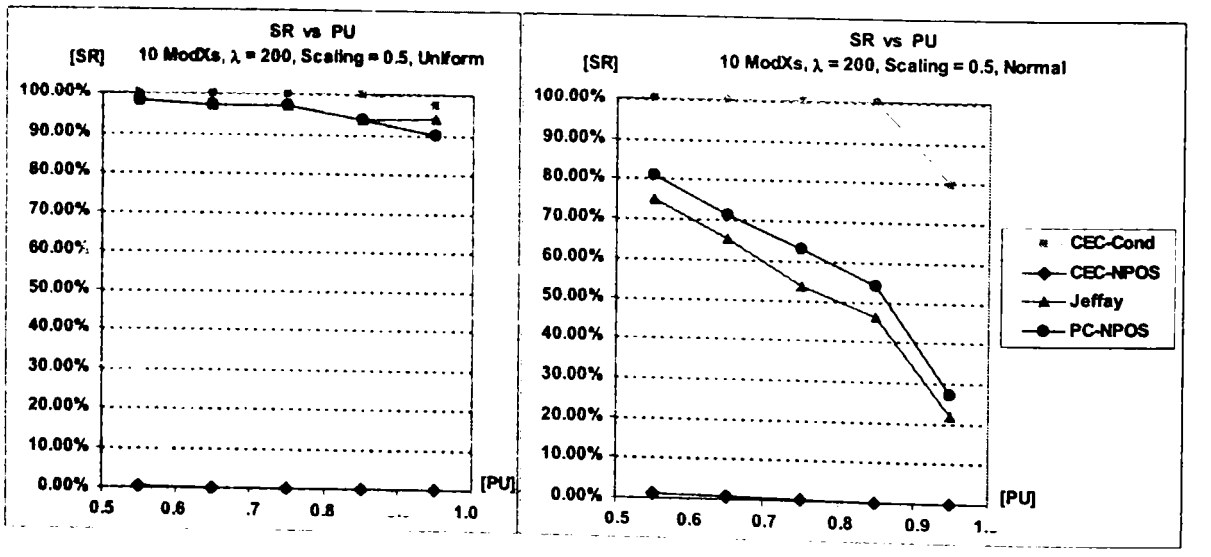

Figura 6-12. *SR* funcţie de *PU*, pentru 10 ModX-uri, $\lambda = 200$ şi factor de scalare 0.5

6.4 Evaluarea algoritmului FENP

Testarea şi analiza performanţelor algoritmului de planificare non-preemptivă a ModX-urilor cu execuţie fixă în cadrul perioadei (FModX-uri), independente, a fost realizată cu ajutorul programului "Fixed", dezvoltat ca şi cele anterioare în mediul Microsoft® VisualStudio™ 6.0 (Visual C++) pentru platforme PC. Configuraţia de operare a programului este de asemenea similară cu cea a celorlaltor unele soft utilizate, fiind specificată printr-un fişier de configurare.

Înainte de aplicarea propriu-zisă a algoritmului de planificare *FENP* asupra seturilor de FModX-uri cu parametrii temporali generaţi aleator, se testează condiţiile de necesitate ale planificabilităţii: (a) testul "Jeffay", pentru evaluări comparative ale comportării algoritmului *FENP* faţă de ceilalţi algoritmi studiaţi, şi (b) condiţia de necesitate *FENP* (Corolarul 5-5):

$$T_{ex}^{M_i} + T_{ex}^{M_j} \le \mathcal{D}_{i,j} = \mathrm{GCD}\left\{T_{pr}^{M_i}, T_{pr}^{M_i}\right\},\ \forall i, j = 1, 2, \ldots, n$$
$$\text{unde: } M_i \equiv \left\{T_{ex}^{M_i}, T_{pr}^{M_i}\right\} \in \boldsymbol{M} = \left\{M_1, M_2, \ldots, M_n\right\} \tag{6-5}$$

Pentru ca programul "Fixed" să poată genera cât mai multe seturi cu şanse de a fi planificate, trebuie ca generarea aleatoare a parametrilor temporali ai FModX-urilor din fiecare set să fie oarecum forţată în sensul apropierii de condiţiile specificate în (6-5):

- Perioadele FModX-urilor, deşi generate aleator, cu distribuţie normală sau uniformă, trebuie să aibă un divizor comun cât mai mare, două câte două;
- Duratele de execuţie, deşi generate aleator, trebuie să fie suficient de mici, comparativ cu divizorul comun al perioadelor, luate două câte două.

Ca urmare, a fost introdus un parametru suplimentar în configurarea sesiunilor de testare aplicate cu programul "Fixed": `GCD_Seed`, care va fi un divizor pentru toate perioadele FModX-urilor, şi de care se ţine cont şi la generarea duratelor de execuţie.

Utilizând programul "Fixed" au fost efectuate peste 90000 de teste, utilizând cele 12 staţii ale reţelei de calculatoare tip PC Pentium III din laboratorul DSPLabs. Testele desfăşurate au urmărit determinarea raportului de planificabilitate *SR* în funcţie de următorii parametri suplimentari (combinaţii ale acestora):

- Numărul total de FModX-uri dintr-un set: `TotMX` = 10, 15, 20;
- Utilizarea procesor (*PU*) implicată de seturile de FModX-uri, încadrată în intervalul: [`MinPU`, `MaxPU`] = [0.2, 0.3], [0.3, 0.4], [0.4, 0.5], [0.5, 0.6], şi [0.6, 0.7];
- Divizorul comun al perioadelor FModX-urilor: `GCD_Seed` = 6, 10, 30 .

Intervalul ce încadrează valorile perioadelor ModX-urilor a fost considerat constant: `IntervTimp` = 510, valoarea minimă pentru perioade fiind 10. Limita maximă pentru intervalul de analiză a planificării a fost configurată pentru T_{LCM} = 1000000000.

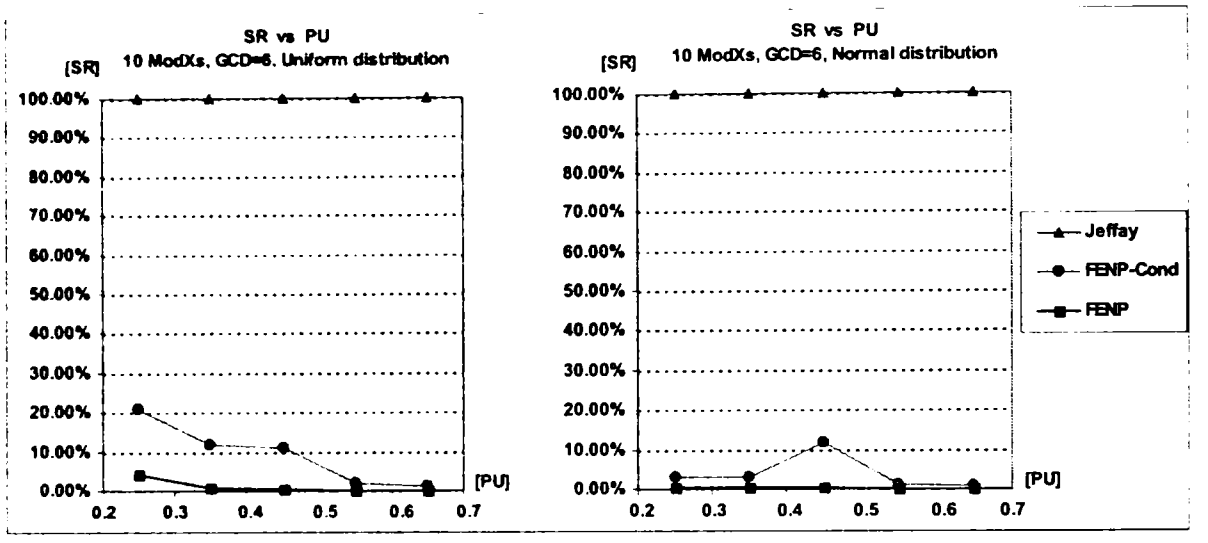

Figura 6-13. *SR* funcţie de *PU*, pentru 10 ModX-uri, cu `GCD_Seed` = 6, distribuţie uniformă (stânga) şi normală (dreapta)

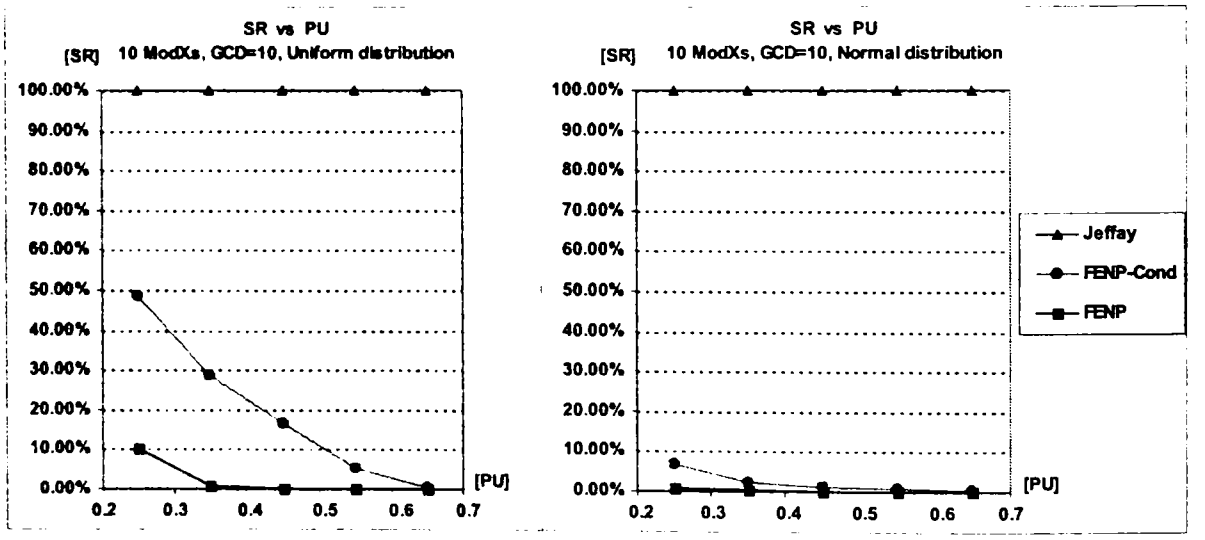

Figura 6-14. *SR* funcţie de *PU*, pentru 10 ModX-uri, cu `GCD_Seed` = 10

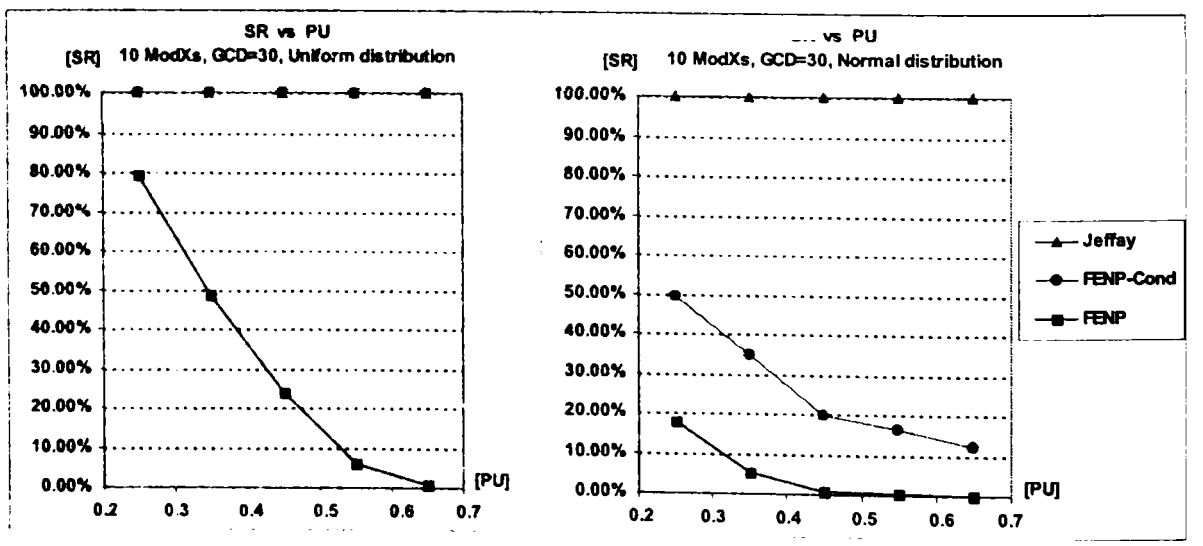

Figura 6-15. *SR* funcţie de *PU*, pentru 10 ModX-uri, cu `GCD_Seed` = 30

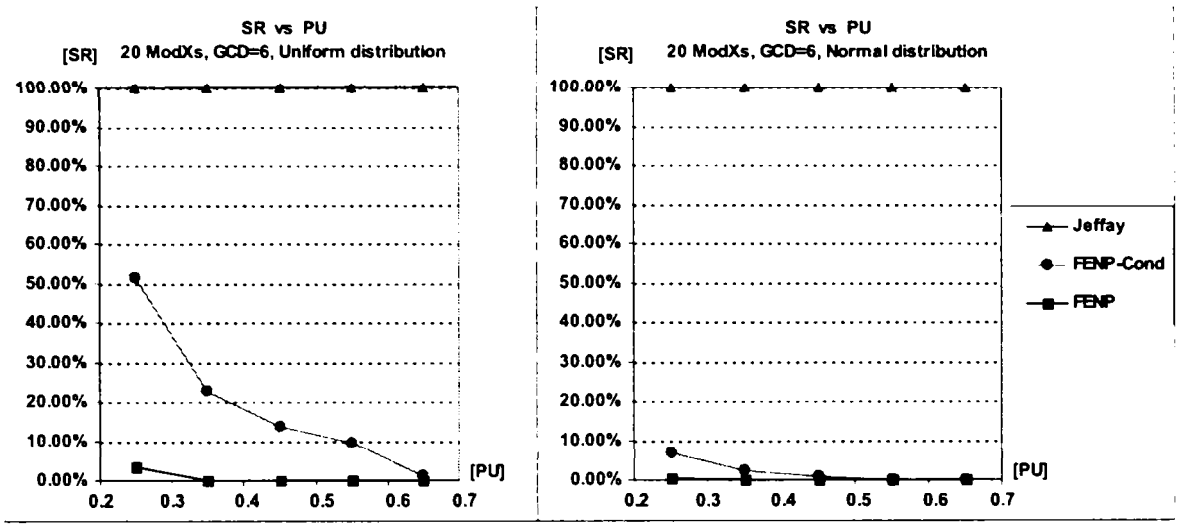

Figura 6-16. *SR* funcție de *PU*, pentru 20 ModX-uri, cu `GCD_Seed` = 6

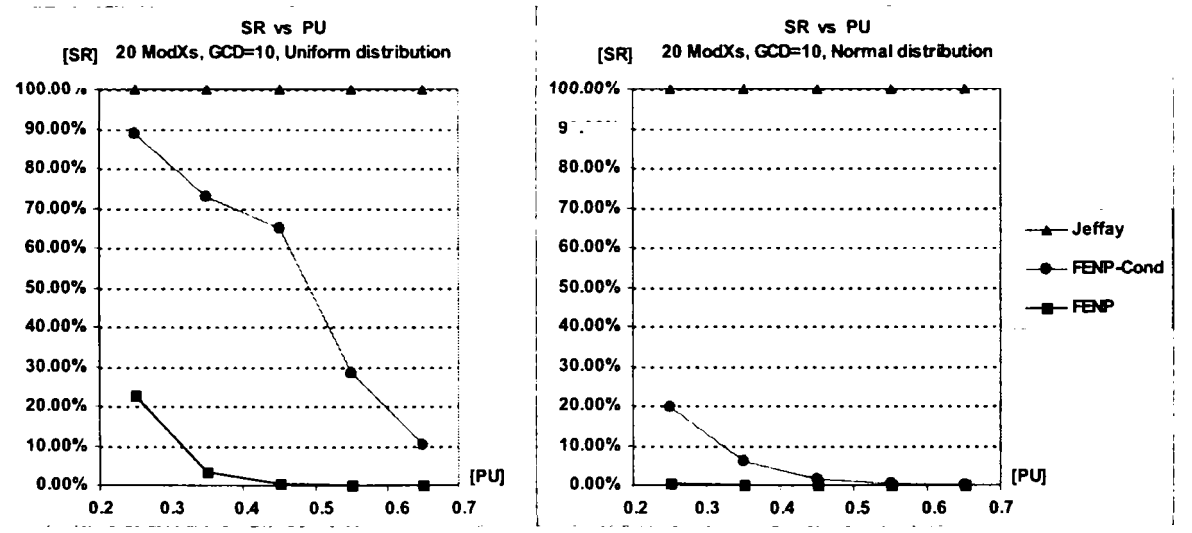

Figura 6-17. *SR* funcție de *PU*, pentru 20 ModX-uri, cu `GCD_Seed` = 10

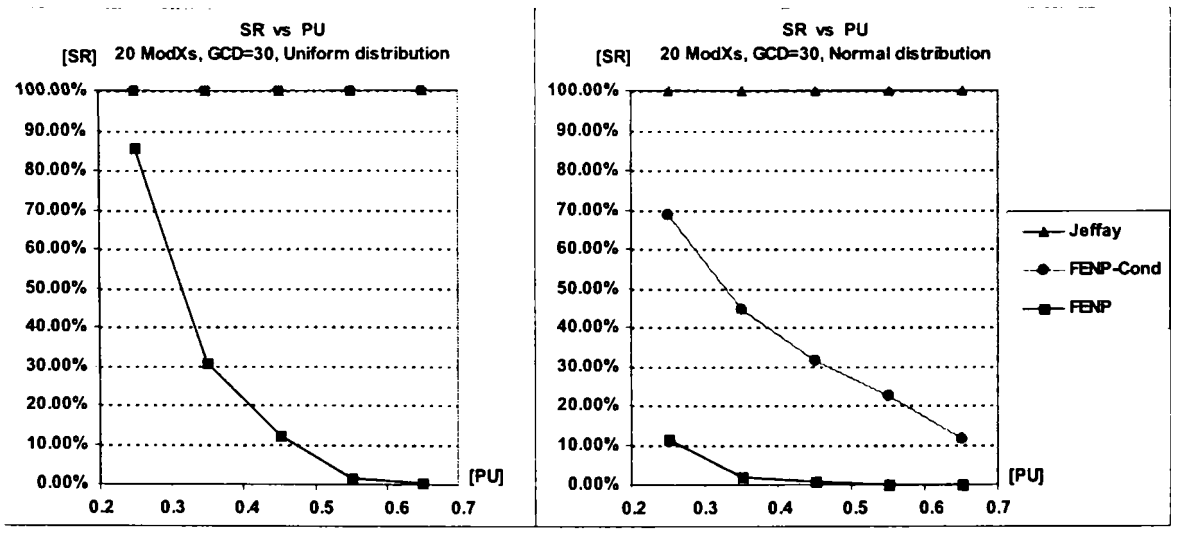

Figura 6-18. *SR* funcție de *PU*, pentru 20 ModX-uri, cu `GCD_Seed` = 30

Rezultatele obţinute cu ajutorul programului "Fixed" pentru seturi de 10 FModX-uri sunt prezentate în Figura 6-13 la Figura 6-15, iar cele pentru seturi de 20 FModX-uri, în Figura 6-16 la Figura 6-18. Din analiza efectuată asupra algoritmului *FENP*, se pot desprinde următoarele observaţii:

- Creşterea valorii divizorului comun al perioadelor FModX-urilor din set măreşte semnificativ rata de succes a planificării *FENP*. Acest lucru este de aşteptat, ţinând cont de condiţia de necesitate *FENP*, (6-5);
- Pentru seturile de FModX-uri cu perioadele generate aleator cu distribuţie uniformă, rata de succes a planificării este mult îmbunătăţită, comparativ cu distribuţia normală. Explicaţia derivă din faptul că într-o distribuţie uniformă, unde valorile perioadelor FModX-urilor "acoperă" întreg intervalul considerat, (a) există o probabilitate mai bună ca divizorul comun al oricărei perechi de perioade să fie relativ mare în comparaţie cu perioadele respective, şi (b) seturile de FModX-uri ale căror perioade au valori "distanţate" una faţă de cealaltă pot fi planificate mai uşor decât în cazul în care perioadele iau valori apropiate şi concentrate într-un subinterval central din domeniul de valori considerat (ca în cazul distribuţiei normale). Această caracteristică a planificării *FENP* vine oarecum în contradicţie cu tipul de distribuţie găsit la parametrii temporali ai task-urilor din aplicaţiile practice, mai apropiat de distribuţia normală decât de cea uniformă;
- Creşterea numărului de FModX-uri din set are ca efect şi creşterea ratei de succes a planificării acestora cu *FENP* (din aceleaşi motive ca cele prezentate la evaluarea algoritmilor *EDFNP* şi *MLFNP*);
- Comparând graficul testului "Jeffay" şi valoarea "de tăiere" a utilizării procesor (*PU* pentru care rata planificabilităţii *SR* se anulează) din figurile de mai sus, cu cele ale celorlalţi algoritmi studiaţi, se poate observa că performanţele planificării *FENP* sunt foarte slabe. Acest fapt este motivat de caracteristicile speciale ale FModX-urilor, în special de restricţia ca execuţia acestora are loc la un acelaşi interval fixat în cadrul perioadei.

6.5 Concluzii

Pentru testarea şi evaluarea performanţelor algoritmilor de planificare non-preemptivă a seturilor de ModX-uri, introduşi şi studiaţi în Capitolul 5, au fost dezvoltate o serie de programe specifice:

- "Evaluation 1" şi "DataAdd", pentru testarea comparativă a algoritmilor *EDFNP* şi *MLFNP*;
- "OnlineScd", pentru testarea comparativă a algoritmilor de planificare online *CEC-NPOS* şi *PC-NPOS*;
- "Fixed" pentru testarea şi evaluarea algoritmului *FENP*.

Procedurile de evaluare a algoritmilor de planificare se bazează pe generarea de seturi de ModX-uri, în manieră aleatoare, cu două din tipurile de distribuţie considerate de bază: distribuţia uniformă şi normală. Metodele de generare ale parametrilor temporali ai ModX-urilor cu cele două tipuri de distribuţie au fost atent

studiate şi implementate. După obţinerea fiecărui set de ModX-uri, au fost aplicate şi contorizate condiţiile de necesitate ale planificării pentru algoritmii studiaţi şi apoi s-a aplicat algoritmul propriu-zis, cu contorizarea rezultatului.

Programele de testare au fost executate în reţeaua de calcul a laboratorului DSPLabs, compusă din 12 staţii PC tip Pentium III, pe o durată totală de peste 16 săptămâni. În acest interval au fost realizate peste 150000 de teste.

Rezultatele testelor efectuate au fost centralizate, analizate şi reprezentate grafic. Diagramele din acest capitol ilustrează câteva din cele mai semnificative rezultate obţinute. Concluziile generale ce se desprind sunt următoarele:

- Algoritmul de planificare non-preemptivă *EDFNP* (Eearliest Deadline First Non-Preemptive) este în mod clar mai eficient ca MLFNP, şi va fi deci preferat în variantele ulterioare de algoritmi de planificare non-preemptivă, care vor trata modele mai complexe de seturi de ModX-uri (de exemplu, ModX-uri cu dependenţe de program, etc.).
- Planificarea în timpul operării sistemului (planificarea online) impune o serie de restricţii pentru task-ul cu acest rol şi pentru întregul set de ModX-uri ce urmează a fi executat în sistem. În mod evident, abordarea planificării online în cicluri periodice (*PC-NPOS*) este de preferat faţă de cea în cicluri cu număr fix de execuţii (*CEC-NPOS*). Din punct de vedere practic, algoritmul *PC-NPOS* este similar cu *EDFNP*, diferenţa constând nu la nivel de algoritm, ci la nivelul setului de task-uri ce urmează a fi planificate: *PC-NPOS* include şi task-ul de planificare online M_S, pe lângă ModX-urile normale.
- Dacă există aplicaţii TR ce necesită generarea sau tratarea de semnale la intervale fixe, exacte de timp, trebuie luat în considerare modelul de FModX, cu algoritmul de planificare corespunzător: *FENP*. Pe de altă parte, includerea acestui algoritm implică o scădere a performanţelor întregului sistem, impunând de asemenea o serie de restricţii suplimentare, relativ severe, pentru parametrii temporali ai task-urilor aplicaţiei.

7 Modelul STR strict pentru aplicaţii critice APNS şi de control încorporat

Capitolul de faţă prezintă unificarea într-un sistem omogen a modelelor dezvoltate în capitolele anterioare pentru reprezentarea timpului, a semnalelor şi a task-urilor TR. Se va introduce un model de STR conceput pentru a oferi un mediu integrat, unitar, de specificare, programare, analiză şi execuţie a aplicaţiilor critice din domenii ca achiziţia şi prelucrarea numerică a semnalelor, orientat pe arhitecturi compacte, de control digital încorporat, cum ar fi de exemplu sistemele bazate pe procesoare numerice de semnal.

7.1 Descrierea generală şi caracteristicile sistemului OPEN-HARTS

Una din observaţiile esenţiale desprinse din studiul sistemelor TR stricte şi a algoritmilor de planificare a ModX-urilor este că faza de analiză offline, apriori execuţiei în sistem a setului de task-uri TR stricte, reprezintă o cerinţă obligatorie. În consecinţă, modelul STR strict conceput cuprinde două elemente principale: un mediu integrat, destinat dezvoltării şi analizei offline a aplicaţiilor şi un nucleu de operare TR strict pe care se încarcă şi se execută aplicaţiile validate.

OPEN-HARTS (Operating Environment for Hard Real-Time Systems) este conceput ca un mediu de operare pentru sisteme şi aplicaţii TR stricte, având două componente majore (Figura 7-1):

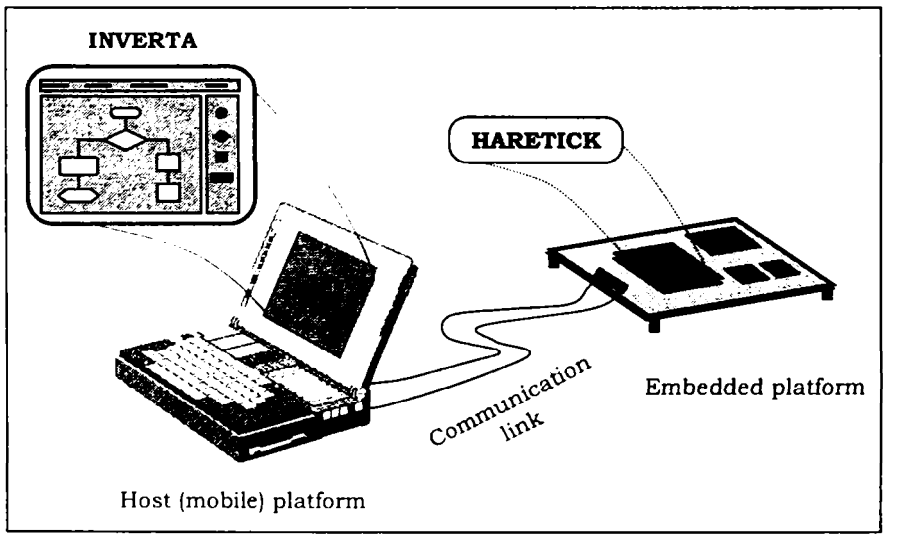

Figura 7-1. Arhitectura generală a sistemului OPEN-HARTS

1) Mediul integrat de dezvoltare şi analiză a aplicaţiilor TR stricte pe sisteme DSP şi de control digital încorporat, INVERTA (Integrated Visual Environment for Real-Time Application Analysis and Development). INVERTA oferă programatorului de sistem şi de aplicaţii TR un mediu vizual integrat, destinat proiectării, specificării şi verificării formale a aplicaţiilor (cu accent pe comportamentul temporal al task-urilor – obligatoriu la cele TR stricte). INVERTA permite de asemenea, programarea funcţională a fiecărui task al aplicaţiei, analiza temporală a task-urilor (extragerea WCET) şi analiza planificabilităţii. Toate operaţiile se realizează în regim "off-line" vis-a-vis de operarea sistemului TR propriu-zis, pe o platformă de calcul ("Host platform"), care poate fi statică (workstation), sau mobilă (Laptop, PDA, Tablet PC, etc.).

2) Nucleul de operare TR hibrid pentru sisteme încorporate, HARETICK (Hard Real-Time Compact Kernel). HARETICK este instalat pe platforma de control digital încorporat ţintă, şi permite încărcarea şi execuţia task-urilor TR stricte (Hard Real-Time tasks) în regim non-preemptiv şi a celor lejere (Soft Real-Time tasks) în regim preemptiv.

Cele două componente ale sistemului OPEN-HARTS sunt interconectate printr-o legătură şi un protocol de comunicaţii, permiţând schimbul de date, comenzi şi informaţii de stare între INVERTA (programatorul de sistem/aplicaţii) şi HARETICK (sistemul TR care operează pe o platformă ţintă).

Întregul sistem se bazează pe o concepţie puternic modulară, atât în ceea ce priveşte arhitectura generală cât şi modul de abordare a proiectării, specificării, analizei şi execuţiei aplicaţiilor TR.

Mediul integrat INVERTA permite programatorului de sistem/aplicaţii dezvoltarea unei aplicaţii TR în regim offline, fără a interfera deci, în această fază, cu sistemul ţintă, pe care poate opera la un moment dat o versiune anterioară a aplicaţiei. Dezvoltarea aplicaţiei începe prin construirea într-un mediu vizual şi într-o manieră modulară, a diagramei de task-uri (ModX-uri şi task-uri TR lejere), rezultând o reţea vizuală, echivalentă cu graful dependenţelor de program (*PDG – Program Dependence Graph*). În continuare, pentru fiecare nod al grafului (reprezentând de fapt un task – ModX sau task TR lejer), se vor specifica următoarele elemente caracteristice:

- Setul parametrilor de intrare şi de ieşire (dependenţa de date faţă de celelalte module).
- Setul variabilelor globale ale aplicaţiei, utilizate de către task.
- Setul semnalelor de intrare şi de ieşire cu care interacţionează task-ul. Pentru fiecare semnal, programatorul poate modela comportarea acestuia în timp, prin intermediul unui set minimal şi complet de parametri temporali (vezi Capitolul 4). Pentru semnalele de intrare, specificarea comportării temporale este obligatorie.
- Prin intermediul unui set minimal şi complet de parametri temporali, programatorul poate modela comportarea în timp a task-urilor TR stricte – ModX-urile (vezi Capitolul 4).

- Comportarea funcţională a fiecărui task se va specifica prin intermediul unui limbaj de programare specific aplicaţiilor TR (limbaj de asamblare, C modificat, etc.).

După proiectarea şi specificarea aplicaţiei, INVERTA va trece în mod automat la faza de verificare/validare a specificaţiilor. Sistemul va utiliza seturi de formule bine stabilite, între parametrii temporali ai semnalelor şi ai task-urilor aplicaţiei (prezentate în Capitolul 4), atât pentru a verifica validitatea specificaţiilor, cât şi pentru a stabili în mod automat valorile parametrilor ce nu au fost specificaţi explicit de programator. INVERTA verifică, de asemenea, dependenţele de control şi de date ale aplicaţiei, ţinând cont şi de specificarea funcţională (codul program) al fiecărui task, şi modifică acolo unde e cazul, afişarea acestor dependenţe în mod corespunzător.

După validarea întregii aplicaţii, atât din punct de vedere funcţional, cât şi al comportamentului în timp, INVERTA va compila codurile sursă ale fiecărui task şi va efectua analiza temporală a programelor, pentru obţinerea timpilor maximi de execuţie ai task-urilor (*WCET*). Tot în cadrul fazei de analiză a aplicaţiei, INVERTA aplică algoritmii planificare special concepuţi pentru modelul de sisteme TR stricte considerate în lucrare (Capitolul 5), pentru a testa planificabilitatea setului de ModX-uri ale aplicaţiei.

În cazul în care INVERTA a găsit o planificare fezabilă a aplicaţiei, se efectuează pregătirile finale pentru încărcarea ei pe platforma ţintă, unde va fi executată sub nucleul de operare HARETICK: link-editare, generarea tabelei de procese (ce conţine şi parametrii temporali ai task-urilor) şi a grafului aplicaţiei, generarea tabelelor de simboluri şi a hărţii de memorie, etc.

Legătura de date şi comenzi (DATALINK) dintre INVERTA şi HARETICK are următoarele caracteristici principale:

- Implementează un protocol de comunicaţii simplu, dar flexibil şi scalabil.
- Permite transmiterea de comenzi utilizator prin intermediul INVERTA, către nucleul de operare HARETICK, care execută o aplicaţie TR oarecare.
- Permite citirea stării de operare a sistemului ţintă, prin comandarea generării de diverse rapoarte de execuţie.
- Permite încărcarea unei aplicaţii noi (sau modificate) în sistemul ţintă, sau descărcarea informaţiilor complete despre o aplicaţie ce se execută pe sistemul ţintă. Datele ce caracterizează execuţia unei aplicaţii TR oarecare pe platforme HARETICK, cuprind printre altele, următoarele elemente:
  - sursele de cod compilat ale task-urilor aplicaţiei;
  - tabela cu descriptorii de procese (incluzând parametrii temporali ai task-urilor);
  - tabela ce descrie graful aplicaţiei;
  - harta de memorie;
  - tabela de simboluri.

Aşadar, prin intermediul legăturii de date, a protocolului de comunicaţie şi a interfeţelor corespunzătoare din INVERTA şi HARETICK, programatorul de aplicaţii poate să citească la orice moment starea sistemului ţintă (în timpul operării acestuia, şi fără a afecta operarea sistemului) – incluzând informaţiile complete

despre aplicaţia executată curent, poate să trimită diverse comenzi utilizator către sistem şi să încarce o nouă aplicaţie pentru a o înlocui pe cea curentă.

La terminarea cu succes a încărcării unei aplicaţii în cadrul sistemului ţintă, nucleul de operare HARETICK va începe execuţia acesteia în timp real. Acest moment defineşte practic începerea operării efective a aplicaţiei în cadrul sistemului, şi predictibilitatea execuţiei, asupra căreia se concentrează materialul din lucrarea de faţă, este garantată începând din acest moment.

HARETICK este un nucleu TR hibrid, multi-tasking şi single-user, pentru sisteme de control încorporat (*embedded systems*), conceput pentru a oferi predictibilitate maximă de execuţie aplicaţiilor TR critice. Caracteristica de nucleu TR hibrid se referă la faptul că HARETICK oferă mecanisme pentru existenţa a două regimuri "simultane" de execuţie: contextul non-preemptiv, destinat execuţiei task-urilor TR stricte (ModX-urile), cu garantarea respectării termenelor specificate (oferind deci, o predictibilitate maximă), şi contextul preemptiv destinat execuţiei task-urilor lejere din punct de vedere al specificaţiilor de comportament temporal. Mecanismele implementate se bazează pe limitarea (sau chiar eliminarea) întreruperilor din sistem, cu excepţia întreruperii de ceas (*RTC – Real-Time Clock*), care are prioritatea maximă.

Observaţie

Modelul nucleului hibrid de operare TR descris mai sus a fost propus şi introdus în contextul implementării HARETICK, ca soluţie (cel puţin parţială) la problema eficienţei scăzute pe care o prezintă sistemele de task-uri ce operează sub planificare non-preemptivă, pe de o parte, şi pe de alta, pentru a compensa pierderile din timpul de operare introduse de analiza de tip "durată maximă de execuţie" (WCET) a task-urilor sistemului.

Conceptele de arhitectură şi implementare introduse în dezvoltarea nucleului TR hibrid HARETICK constituie subiectul secţiunii următoare.

7.2 Principiile de operare ale sistemului propus

Principiile de operare ale sistemului OPEN-HARTS au fost prezentate în esenţă în paragraful anterior. Pentru o exemplificare şi mai concretă a capabilităţilor sistemului, vom considera un scenariu ipotetic de utilizare a OPEN-HARTS.

Presupunem că, la un moment dat, există un sistem digital încorporat cu rolul de control al unui echipament cu cerinţe critice de operare. Pe platforma sistemului rulează nucleul HARETICK, ce execută o anume aplicaţie TR. Programatorul sistem doreşte modificarea aplicaţiei care rulează în mod curent pe platforma ţintă, mai precis, doreşte adăugarea a încă unui ModX în aplicaţie, de exemplu. Secvenţa de paşi şi evenimente ce vor apărea în acest scenariu, se desfăşoară după cum urmează:

1. Programatorul sistem conectează sistemul de calcul propriu (presupunem că e vorba de un sistem portabil, tip Laptop), pe care este instalat mediul de dezvoltare INVERTA, la platforma ţintă, prin intermediul legăturii DATALINK.

2. Din mediul INVERTA, utilizatorul selectează opțiunea de generare a rapoartelor de stare, care este trimisă ca o comandă sistem către HARETICK, prin intermediul legăturii de comunicație.

3. Comanda de generare a rapoartelor de stare este recepționată de nucleul HARETICK de pe sistemul țintă, interpretată şi executată (în regim TR lejer). Ca urmare, HARETICK transmite informația completă de stare curentă către INVERTA. Informația de stare cuprinde elemente legate de aplicația ce se execută curent pe sistemul țintă (tabela cu descriptorii de procese, graful aplicației, sursele de cod ale task-urilor, etc.), cât şi elemente legate caracteristicile temporale ale componentelor de bază ale nucleului (dispatcher, online scheduler, etc.), şi timpii şi evenimentele desfăşurate în cadrul operării sistemului.

4. Informația de stare recepționată este interpretată în cadrul mediului INVERTA, rezultând, printre altele, o afişare sub formă de graf vizual a aplicației ce rulează pe sistemul țintă, împreună cu valorile tuturor parametrilor acestora şi cu codul program (specificarea funcțională a task-urilor).

5. Programatorul de aplicație poate în acest moment să modifice aplicația respectivă (prin adăugarea unui nou ModX de exemplu, aşa cum am presupus în scenariu).

6. După adăugarea ModX-ului nou în cadrul aplicației, şi după specificarea lui completă (stabilirea valorilor pentru parametrii temporali, scrierea codului program, stabilirea legăturilor de control şi date cu celelalte task-uri, etc.), mediul INVERTA inițiază în mod automat verificarea şi validarea întregii aplicații.

7. În momentul în care specificațiile aplicației au fost validate de către INVERTA, se activează faza de analiză temporală a programului (compilarea codului noului ModX, calculul WCET), şi analiză a planificabilității setului de ModX-uri.

8. Găsirea unei planificări fezabile a noii aplicații şi pregătirile finale pentru încărcarea acesteia în sistemul țintă, încheie setul de operații efectuate în mod offline de către INVERTA.

   Ca observație, menționăm că toate operațiile offline pe care le desfăşoară mediul integrat de dezvoltare INVERTA (paşii 4 – 8) pot fi efectuate şi cu platforma gazdă (laptop-ul programatorului de sistem) deconectată de la sistemul țintă, pe care încă rulează aplicația curentă. De asemenea, timpul nu este o coordonată esențială pentru efectuarea acestor operații.

   În cazul în care calculatorul gazdă a fost deconectat (şi deplasat) de la sistemul țintă, va trebui reconectat la acesta, pentru etapele următoare.

9. Noua aplicație, dezvoltată şi analizată cu ajutorul sistemului INVERTA este încărcată pe sistemul țintă, prin intermediul unor comenzi specifice, transmise către HARETICK.

10. Întreaga procedură se termină în momentul încărcării cu succes pe platforma ţintă a tuturor datelor necesare planificării şi executării online a noii aplicaţii. După acest moment, aplicaţia va opera în timp-real, cu garantarea respectării tuturor specificaţiilor temporale pentru task-urile TR stricte (ModX-urile aplicaţiei).

De remarcat faptul că, până la terminarea pasului 10 din secvenţa de mai sus, cu alte cuvinte, până când noua aplicaţie este complet încărcată în sistemul ţintă, acesta va continua să execute aplicaţia veche. Comutarea execuţiei dintre cele două aplicaţii existente în sistem (echivalentă cu o sincopă în operarea sistemului) se face într-o manieră controlată şi predictibilă. Această întrerupere a operării este singurul eveniment care poate perturba interacţiunea sistemului cu mediul înconjurător.

SECŢIUNEA III. CONTRIBUŢII LA IMPLEMENTAREA ŞI EVALUAREA *STR* STRICTE

În capitolele secţiunii de faţă sunt prezentate detaliile de implementare ale unui sistem de operare (nucleu) timp-real cu caracteristici care îi permit garantarea respectării termenelor de timp pentru task-urile critice ale aplicaţiilor de achiziţie şi prelucrare numerică de semnal sau de control încorporat.

Astfel, nucleul de operare HARETICK ("Hard Real-Time Compact Kernel") reprezintă un studiu de caz, complet şi consistent, pentru implementarea practică, testarea şi evaluarea principiilor, modelelor teoretice şi algoritmilor dezvoltaţi şi discutaţi în cadrul secţiunii anterioare, cu privire la specificarea, analiza şi planificarea aplicaţiilor critice.

Nucleul HARETICK a fost parţial implementat pe platforma de prelucrare numerică de semnal Motorola DSP56307EVM, astfel încât să se poată testa şi evalua principalele componente funcţionale ale nucleului şi să se poată verifica dacă operarea acestuia corespunde îndeosebi modelelor teoretice dezvoltate pentru clasa task-urilor cu specificaţii stricte de comportare temporală (HRT – Hard Real-Time) ale aplicaţiilor critice.

8 Nucleul de operare HARETICK: caracteristici generale, componente şi relaţia dintre acestea

8.1 Caracteristici generale

Conceperea, proiectarea şi implementarea nucleului HARETICK are la bază faptul că sunt vizate sistemele de control digital încorporat (embedded systems). Prin urmare, HARETICK este un nucleu de operare timp-real de tip *single-user*: programatorul de sistem/aplicaţii poartă întreaga responsabilitate a proiectării, programării şi utilizării sistemului.

De asemenea, HARETICK este un nucleu *mutlitasking*, permiţând execuţia concurentă, planificată a unor seturi compuse din mai multe task-uri. Nucleul oferă facilităţi pentru operarea cu predictibilitate maximă a aplicaţiilor critice, garantând respectarea termenelor stricte de timp impuse de acestea. În continuare vom prezenta o serie de alte caracteristici şi proprietăţi importante ce definesc arhitectura şi operarea nucleului HARETICK.

Arhitectura modulară

Sistemul HARETICK este conceput şi proiectat într-o manieră puternic modularizată, vizându-se simplitatea şi reducerea pe cât posibil a dimensiunilor acestuia. Pentru contextul strict de execuţie a task-urilor, sistemul nu efectuează nici un fel de analize de planificabilitate şi fezabilitate, ci implementează algoritmi de planificare, cu comportare temporală cunoscută, asupra aplicaţiei analizate şi optimizate apriori, offline.

Componentele software cu specificaţii temporale stricte (HRT) ale sistemului şi ale aplicaţiilor dezvoltate pentru acesta, sunt proiectate şi implementate sub formă de module software independente, care comunică între ele prin intermediul parametrilor de intrare/ieşire şi al variabilelor globale.

Task-urile existente la un moment dat în cadrul HARETICK sunt împărţite în două categorii: task-uri sistem, care definesc practic nucleul de operare şi sunt încărcate în faza de iniţializare a sistemului, şi task-uri aplicaţie, care sunt încărcate la un moment dat în sistem. Nucleul HARETICK permite încărcarea, descărcarea sau înlocuirea unei aplicaţii TR la orice moment, prin intermediul unor comenzi operator specifice.

Gestionarea timpului

Timpul, ca parametru esenţial de operare al sistemului, este reprezentat şi gestionat cu ajutorul unor structuri şi mecanisme dedicate. *Baza de timp* a sistemului

este implementată de *ceasul timp-real* (RTC) – elementul activ de gestionare a timpului în HARETICK. RTC are două roluri principale: (i) contor de timp, utilizat pentru referiri ale componentelor sistemului la instanța curentă de timp, şi (ii) singura sursă (predictibilă) de întreruperi. *RTC este o sursă predictibilă de întreruperi în sensul că se poate determina oricând, şi de către orice task, momentul apariţiei următoarei întreruperi.*

Pe lângă RTC, nucleul HARETICK gestionează încă două structuri de timp. *Timpul absolut sistem*, este utilizat pentru referiri ale operatorului şi ale task-urilor din sistem la timpul absolut, şi poate fi setat (modificat) de către operatorul sistemului în orice moment, prin intermediul unor comenzi specifice. *Timpul de planificare* este utilizat şi gestionat de către planificatorul contextului strict de execuţie al sistemului pentru calcularea şi specificarea momentelor la care sunt planificate execuţiile task-urilor TR stricte.

Spre deosebire de timpul absolut sistem, celelalte două structuri – ceasul timp-real şi timpul de planificare, lucrează practic în domeniu temporal relativ. Pe de altă parte, toate cele trei tipuri de timp cu care lucrează nucleul HARETICK sunt sincronizate între ele.

Gestionarea proceselor

Procesele (task-urile) cu care operează sistemul HARETICK sunt organizate, reprezentate şi gestionate în manieră modulară, utilizând structuri de memorie şi mecanisme specifice pentru fiecare task în parte: zonă dedicată pentru cod, zonă pentru parametrii de ieşire, zonă pentru variabilele globale, zonă comună pentru parametrii de intrare şi variabilele locale, utilizată de către task-ul aflat în execuţie curentă, tabela cu graful aplicaţiei, tabela cu identificatorii de procese, tabela de simboluri, etc.

Task-urile cu care operează nucleul HARETICK sunt împărţite în două categorii principale: (1) task-uri TR stricte (task-uri HRT – Hard Real-Time) denumite ModX-uri (Module eXecutabile), şi (2) task-uri lejere (task-uri SRT – Soft Real-Time). În consecinţă, sistemul HARETICK are două contexte de execuţie complementare: contextul strict (sau contextul HRT) în care ModX-urile se execută în regim non-preemptiv, şi contextul lejer (sau contextul SRT), pentru execuţia task-urilor lejere în regim preemptiv.

Din acest punct de vedere, HARETICK este un *nucleu de operare hibrid*, comutarea celor două contexte de execuţie fiind realizată de un task sistem special – executivul TR al HARETICK.

Gestionarea memoriei

Principalul deziderat urmărit la proiectarea şi implementarea sistemului HARETICK este operarea predictibilă, cu garantarea respectării termenelor stricte de timp pentru task-urile care au aceste cerinţe. Prin urmare toate structurile şi mecanismele implementate de nucleu au o arhitectură şi o comportare statică, bine determinată şi limitată în timp şi spaţiu.

Memoria este gestionată în mod liniar de către nucleu, şi este împărţită în două secţiuni – memoria sistem şi memoria aplicaţie. Cu toate acestea, accesul task-urilor

la toate zonele de memorie este permis, nefiind implementat nici un mecanism care să le restricţioneze şi să definească zone de memorie "protejate". Motivaţia abordării este dată de caracteristica principală pentru care este proiectat sistemul: nucleu de operare TR pentru sisteme încorporate (embedded systems), în cadrul căruia programatorul/operatorul are întregul control – şi de asemenea, întreaga responsabilitate.

Mecanismele IPC

Comunicaţia dintre task-uri (IPC – Inter-Process Communication) este realizată prin intermediul parametrilor de intrare/ieşire şi al variabilelor globale. ModX-urile, ca task-uri stricte, periodice, care se execută în context non-preemptiv, implementează acţiuni de tip atomic.

Astfel, se elimină necesitatea implementării de tehnici pentru sincronizarea şi controlul acceselor concurente la resursele partajate ale nucleului (inclusiv la variabilele globale). De asemenea, sunt evitate problemele de planificare de tip inversare a priorităţii.

Interfaţa cu utilizatorul

HARETICK pune la dispoziţia utilizatorului/operatorului o interfaţă cu sistemul. Interfaţa este realizată printr-o serie de componente sistem care implementează un protocol simplu şi flexibil de comunicaţii de date, un protocol de comandă şi interogare a stării sistemului, şi un set de proceduri de încărcare a unei aplicaţii noi în sistem.

8.2 Componentele HARETICK

Arhitectura generală a nucleului HARETICK (vezi Figura 8-1) este proiectată ţinând cont de caracteristica esenţială a operării sistemului – existenţa celor două contexte de operare: contextul lejer (preemptiv) şi contextul strict (non-preemptiv). În consecinţă, toate elementele sistemului (task-urile sistem şi ale aplicaţiei) se vor clasifica după operarea într-unul din cele două contexte. Sistemul va cuprinde aşadar, ModX-uri (task-uri TR stricte) şi task-uri lejere.

În continuare vom prezenta pe scurt componentele principale ale HARETICK, specificând tipul lor (task-uri lejere sau ModX-uri) şi funcţiile pe care le îndeplinesc. O reprezentare intuitivă a componentelor HARETICK, precum şi a relaţiilor stabilite între acestea, este redată în Figura 8-1. Modulele în stadiu de implementare completă sunt reprezentate cu blocuri complet colorate, cele haşurate sunt parţial implementate (funcţionalitate de bază), iar cele albe urmează a fi dezvoltate în continuare.

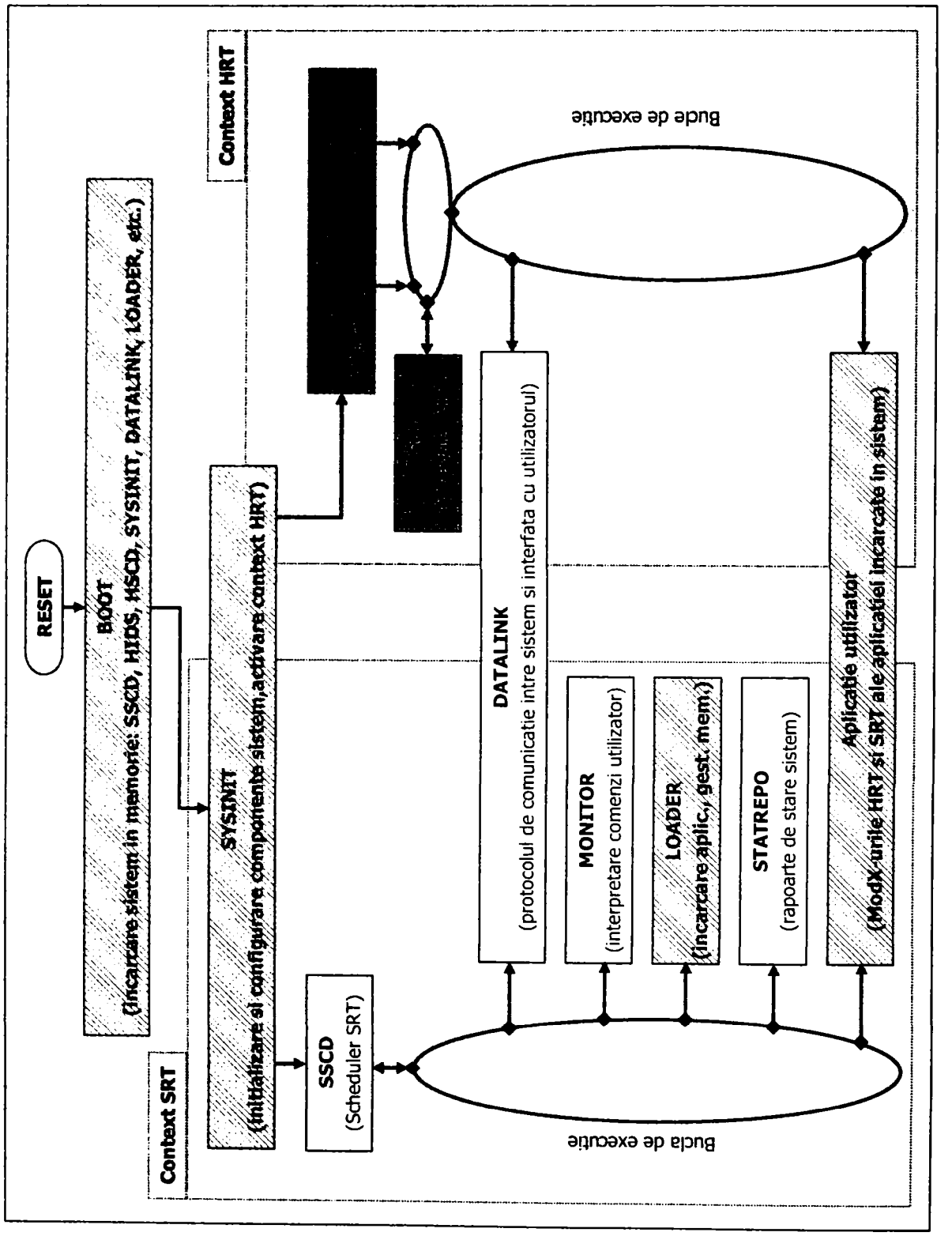

Figura 8-1. Componentele principale HARETICK şi relația dintre ele

8.2.1 Încărcarea şi lansarea sistemului: BOOT

Secvenţa (task-ul) de încărcare şi startare a sistemului HARETICK este denumită "BOOT". Are rolul de a încărca în memorie structurile de date şi codul tuturor componentelor de bază ale HARETICK: SYSINIT, SSCD, HDIS, TiLT, HSCD, DATALINK, MONITOR, LOADER, STATREPO, etc.

Task-ul BOOT este apelat şi executat de către procesorul sistemului în momentul alimentării sau după o operaţie de RESET, şi se găseşte înscris în memoria EPROM internă sau externă a procesorului ţintă.

Task-ul BOOT nu poate fi clasificat nici ca task lejer, nici ca ModX, deoarece în această etapă de operare a HARETICK, încă nu au fost iniţiate cele două contexte de operare.

8.2.2 Iniţializarea sistemului: SYSINIT

Task-ul de iniţializare a sistemului este denumit "SYSINIT". Este task-ul lansat în execuţie de către BOOT, după ce acesta a încărcat cu succes în memorie sistemul.

Rolul lui SYSINIT este de a iniţializa parametrii de operare ai sistemului: variabilele globale sistem, variabilele şi parametrii celorlalte componente, cum ar fi pointerii către tabela cu descriptorii de procese şi tabela de dispatch, etc. După efectuarea iniţializărilor, SYSINIT activează contextul strict de execuţie prin startarea ceasului TR (*RTC – Real-Time Clock*), care în mod automat, prin mecanismul întreruperilor de timer, apelează executivul sistemului, HDIS (Hard real-time Dispatcher).

SYSINIT este un task de tip lejer, deci practic, se poate considera că, odată cu lansarea în execuţie a lui SYSINIT de către BOOT, se activează contextul lejer de execuţie. La finalul secvenţei lui SYSINIT este lansat planificatorul task-urilor lejere, SSCD, care gestionează în continuare operarea acestora în contextul lejer de execuţie.

8.2.3 Comunicaţia de date: DATALINK

Comunicaţia de comenzi şi date dintre nucleul HARETICK şi mediul de dezvoltare INVERTA de pe un calculator gazdă este realizată cu ajutorul setului de task-uri DATALINK, care implementează o interfaţă simplă şi flexibilă de comunicaţii bidirecţionale cu următoarele caracteristici:

- (→) Încărcarea de comenzi utilizator, ce urmează apoi a fi interpretate şi executate de către HARETICK;
- (→) Încărcarea de aplicaţii şi date în sistem;
- (←) Descărcarea de rapoarte legate de starea curentă de execuţie a sistemului;
- (←) Descărcarea periodică de rapoarte de timp.

În notaţiile de mai sus a fost folosit simbolul (→) pentru a indica un sens al comunicaţiei dinspre calculatorul gazdă către platforma ţintă (cu HARETICK), iar (←) pentru sensul invers.

Grupul de task-uri DATALINK este compus atât din ModX-uri, cât şi din task-uri lejere (vezi şi Figura 8-1). ModX-urile DATALINK sunt responsabile pentru implementarea operaţiilor de transmisie/recepţie a biţilor de date pe interfaţa de comunicaţii a sistemului. Aceste operaţii sunt de natură periodică şi strictă, trebuind să respecte o rată (tact de bit) bine stabilită. ModX-urile DATALINK se vor executa aşadar în contextul strict, asigurând respectarea debitului stabilit pentru comunicaţie (respectarea ratei de comunicaţie). Ele vor fi planificate şi executate în mod continuu, periodic, chiar dacă, la un moment dat, nu există informaţie ce trebuie transmisă/recepţionată.

Task-urile lejere DATALINK au ca scop prelucrarea la un nivel mai superior a informaţiilor vehiculate pe legătura de date a sistemului. Ele utilizează zone de memorie tampon (buffere) de transmisie/recepţie pentru a dezasambla/asambla cuvintele de date vehiculate în timpul comunicaţiilor. Task-urile lejere DATALINK se vor executa în contextul corespunzător, putând fi astfel întrerupte de către ModX-urile ce rulează în cadrul sistemului. Execuţia task-urilor lejere DATALINK nu prezintă un tipar periodic şi nici specificaţii stricte, ele fiind apelate de către alte task-uri sistem ce necesită la un moment dat utilizarea legăturii de date cu calculatorul gazdă.

8.2.4 Interfaţarea cu operatorul: MONITOR

Task-ul MONITOR are rolul de interpretor de comenzi utilizator, transmise de către mediul de dezvoltare INVERTA de pe calculatorul gazdă, prin intermediul interfeţei de comunicaţii.

De asemenea, MONITOR determină lansarea acţiunilor de încărcare a unei aplicaţii, descărcarea raportului de stare sistem, etc., prin invocarea task-urilor sistem corespunzătoare.

MONITOR este un task de tip lejer, implementând interacţiuni cu utilizatorul ce nu sunt critice pentru operarea sistemului ţintă.

8.2.5 Încărcarea unei aplicaţii: LOADER

Încărcarea unei aplicaţii în sistemul ţintă este realizată prin intermediul task-ului LOADER, care este invocat de către MONITOR ca urmare a interpretării comenzii aferente primite de la operator. Utilizând interfaţa de comunicaţii DATALINK, task-ul LOADER recepţionează de la mediul de dezvoltare INVERTA de pe calculatorul gazdă toate datele şi secvenţele de cod ce definesc noua aplicaţie, printre care:

- tabela cu identificatorii de proces (PDT – Process Descriptor Table) corespunzătoare aplicaţiei, în care se regăsesc principalii parametri ai task-urilor, incluzând pe cei ce descriu comportarea în timp (e.g. perioada, intervalul limită, durata maximă de execuţie – WCET, contorul de execuţii, şi aşa mai departe);
- tabela graful direcţionat aciclic de program al aplicaţiei (PDAGT – Program Directed Acyclic Graph Table) ce descrie dependenţele de control şi de date ale task-urilor;
- codul task-urilor aplicaţiei;

- listele parametrilor de intrare, de ieşire şi a celor globali pentru fiecare task;
- alte atribute ale task-urilor.

După recepţionarea aplicaţiei prin DATALINK, LOADER creează structurile de date şi tabelele aferente, şi configurează memoria sistemului prevăzută pentru încărcarea aplicaţiei:

- înscrie codul fiecărui task în zona de memorie corespunzătoare codului aplicaţiei (zona de text a aplicaţiei, "CODE");
- creează zonele de memorie corespunzătoare parametrilor de ieşire ai task-urilor ("Data Out");
- generează tabela de simboluri a aplicaţiei, în care se face corespondenţa dintre numele fiecărui parametru de intrare sau ieşire al task-urilor, cu adresa din memorie unde este localizat; etc.

LOADER are aşadar şi rolul de gestionare a memoriei, respectiv a structurilor din memoria sistemului ţintă, rezervate aplicaţiilor. Datorită faptului că HARETICK implementează un nucleu de operare pentru sisteme încorporate (embedded systems), pe de o parte zona structurilor de memorie proprii nucleului este gestionată practic de elementele acestuia şi este configurată din etapa de proiectare şi implementare, iar pe de altă parte nu există mecanisme care să controleze/restricţioneze accesul task-urilor la diversele zone de memorie – acest lucru fiind responsabilitatea programatorului de sistem/aplicaţie.

La finalul secvenţei LOADER, aplicaţia încărcată cu succes este lansată în execuţie. Task-urile lejere ale aplicaţiei vor fi planificate şi activate de către planificatorul contextului lejer de execuţie (SSCD), iar ModX-urile aplicaţiei vor fi planificate de către planificatorul contextului strict de execuţie (HSCD), fiind activate de către executivul sistemului (HDIS). Momentul lansării în execuţie a aplicaţiei este considerat ca fiind startul operării TR a aplicaţiei în cadrul sistemului.

LOADER este un task de tip lejer, întregul set de operaţii, de la activarea şi până la lansarea acestuia în execuţie fiind considerat în afara contextului de operare timp-real a sistemului. Până la finalizarea încărcării şi configurării noii aplicaţii în cadrul sistemului, HARETICK continuă să execute vechea aplicaţie, în regim timp-real ce respectă toţi parametrii de comportare temporală specificaţi pentru aceasta.

8.2.6 Planificatorul task-urilor TR lejere: SSCD

Task-ul SSCD (Soft real-time Scheduler) are rolul de planificare a task-urilor lejere ale sistemului şi aplicaţiei, pentru execuţie în context lejer pe platforma HARETICK.

SSCD este la rândul său un task lejer, execuţia sa fiind întreruptă când este cazul de către ModX-urile aparţinând contextului strict al sistemului.

8.2.7 Planificatorul ModX-urilor: HSCD

Task-ul HSCD (Hard real-time Scheduler) are rolul de planificare a task-urilor TR stricte (ModX-uri) ale sistemului şi aplicaţiei încărcate pe acesta, pentru execuţie în contextul strict al HARETICK.

HSCD gestionează o tabelă de dimensiuni fixe (Tabela de Dispatch) în care înregistrează pe rând, ModX-urile ce urmează a fi executate în ciclul curent de planificare în contextul strict al HARETICK. O înregistrare din tabelă conţine identificatorul ModX-ului corespunzător şi instanţa timpului sistem (valoarea RTC – ceasul timp-real) la care este planificată execuţia acestuia.

Un ciclu de planificare este definit prin intervalul de timp corespunzător a două execuţii consecutive ale HSCD. Astfel, Tabela de Dispatch conţine la orice moment, planificările ModX-urilor ce constituie ciclul curent de planificare.

Pe lângă înregistrările normale, pentru ModX-urile sistem, Tabela de Dispatch conţine o înregistrare specială, ce identifică pe HSCD.

HSCD este un ModX, fiind astfel executat de către HARETICK în contextul strict de operare.

8.2.8 Comutarea execuţiei ModX-urilor: HDIS

Comutarea execuţiei ModX-urilor în cadrul contextului strict al HARETICK este gestionată de către executivul TR al sistemului, HDIS (Hard real-time Dispatcher).

HDIS utilizează Tabela de Dispatch gestionată de HSCD, pe care o accesează în citire, înregistrare cu înregistrare. Accesarea a tabelei se face în mod circular, astfel încât după lansarea în execuţie a ultimului ModX din ciclul curent de planificare, HDIS va reveni la baza tabelei, unde găseşte înregistrarea corespunzătoare primei planificări din următorul ciclu.

Un alt rol important al HDIS este acela de salvare/restaurare a contextului task-urilor lejere a căror execuţie este întreruptă de către contextul strict de operare a nucleului.

HDIS este apelat în mod automat în cadrul sistemului HARETICK, de către întreruperea de ceas timp-real (RTC), la momentul de timp ce specifică planificarea unui ModX. Odată activat, HDIS efectuează următoarele acţiuni:

- verifică dacă a fost întrerupt contextul lejer. În caz afirmativ, se efectuează salvarea contextului de execuţie lejeră a task-ului întrerupt într-o zonă dedicată;
- identifică ModX-ul ce urmează a fi lansat în execuţie la momentul curent, din Tabela de Dispatch;
- citeşte următoarea înregistrare din Tabela de Dispatch, corespunzând ModX-ului ce urmează ca planificare celui curent, şi extrage momentul de start al execuţiei acestuia;
- programează timerele ceasului de timp-real (RTC) pentru a genera întrerupere la momentul de start al ModX-ului următor;
- decrementează contorul de execuţii al ModX-ului curent (dacă acesta are valoare finită);
- trece ModX-ul curent din starea "planificat" ("SCD"), fie în starea "de execuţie" ("RUN") – dacă valoarea contorului de execuţie era nenulă înaintea decrementării, fie în starea " ModX fantomă" ("GST");
- dacă ModX-ul curent este în starea "RUN", îi cedează procesorul şi îl apelează pentru execuţie.

La terminarea execuției ModX-ului curent, acesta apelează din nou HDIS. În această situație HDIS efectuează următoarele acțiuni:

- trece ModX-ul curent din starea "de execuție" (RUN) sau "ModX fantomă" (GST), înapoi în "planificat" (SCD) sau în "gata de planificare" (RDY);
- avansează referința (pointerul) de citire al Tabelei de Dispatch către ModX-ul următor;
- verifică dacă există suficient timp pentru a comuta contextul strict cu cel lejer, permițând execuția task-urilor lejere ale sistemului şi aplicației. Dacă nu este timp suficient, HDIS va iniția o buclă infinită în care va aştepta apariția întreruperii RTC corespunzătoare planificării ModX-ului următor. Dacă se decide comutarea controlului către contextul lejer, se va restaura contextul task-ului lejer a cărui execuție a fost întreruptă.

Din cele de mai sus rezultă că HDIS are două componente: HDIS Prefix (HDIS_PRE) care este apelat de întreruperea RTC şi lansează în execuție ModX-uri, şi HDIS Suffix (HDIS_SUF) care este apelat la terminarea fiecărui ModX.

HDIS este, la rândul său, un ModX, fiind astfel executat în context strict în cadrul HARETICK.

8.2.9 Raportarea stării curente a sistemului: STATREPO

Task-ul de raportare a stării curente a sistemului este denumit STATREPO. Rolul acestuia este de a trimite către interfața utilizator de pe calculatorul gazdă (INVERTA), rapoarte la cerere, utilizând interfața de comunicație DATALINK. Conținutul rapoartelor generate de STATREPO cuprinde:

- momentul curent al generării raportului;
- tabela cu identificatorii de proces (PDT – Process Descriptor Table) corespunzătoare aplicației executate curent, în care se regăsesc principalii parametri ai task-urilor, incluzând pe cei ce descriu comportarea în timp (e.g. perioada, intervalul limită, durata maximă de execuție – WCET, contorul de execuții, şi aşa mai departe);
- graful dependențelor de program (PDG – Program Time Graph) ale task-urilor aplicației (descrie, cu ajutorul unei structuri unice de graf aciclic, direcționat, dependențele de control şi de date);
- codul task-urilor aplicației;
- listele parametrilor de intrare, de ieşire şi a celor globali pentru fiecare task;
- alte atribute ale task-urilor;
- harta memoriei;
- conținutul curent al Tabelei de Dispatch;
- tabela de simboluri a aplicației; etc.

Aşa cum a fost descris, în cadrul referatului precedent, sistemul OPEN-HARTS împreună cu componenta sa dedicată dezvoltării şi analizei aplicațiilor TR stricte INVERTA, informațiile legate de starea curentă a sistemului HARETICK şi a aplicației ce rulează pe acesta sunt necesare mediului INVERTA pentru analiza unei noi aplicații sau pentru modificarea celei curente.

STATREPO mai poate fi configurat pentru a genera şi transmite rapoarte periodice privind timpii importanți ai execuției sistemului (istoria execuției).

STATREPO este un task de tip lejer, fiind activat doar la cererea operatorului şi putând fi întrerupt de către contextul strict de execuţie al HARETICK de câte ori este necesar.

8.2.10 Subrutina de raportare a evenimentelor sistem: TiLT

TiLT ("Time Log Tool") este o subrutină aparţinând task-ului TR strict (ModX-ului) HDIS. TiLT este apelată atât de HDIS_PRE înainte de lansarea în execuţie a unui ModX, cât şi de HDIS_SUF la terminarea execuţiei ModX-ului curent.

Rolul TiLT este de a stoca într-o zonă dedicată de memorie tampon circulară (buffer circular), informaţii legate de evenimentele lansării sau terminării execuţiei ModX-urilor sistemului:

- valoarea ceasului timp-real (RTC) în momentul apariţiei evenimentului;
- starea HDIS din acel moment;
- valoarea contorului de execuţie.

Fiind o subrutină a ModX-ului HDIS, TiLT este executat, la rândul său, în context strict, non-preemptiv.

8.3 Relaţia dintre componentele nucleului HARETICK

Figura 8-1 reprezintă intuitiv atât componentele de bază ale nucleului TR HARETICK, cât şi relaţia dintre acestea. În figură sunt evidenţiate cele două contexte de execuţie ce constituie caracteristica distinctivă a HARETICK: contextul strict (HRT) şi contextul lejer (SRT).

Relaţia dintre componentele HARETICK poate fi prezentată şi prin scenariul ce presupune la un moment dat pornirea platformei ţintă (sistemul de control încorporat) ce operează sub nucleul HARETICK, după care se încarcă în sistem o aplicaţie oarecare pentru a fi executată. Scenariul evidenţiază următoarele evenimente:

1. În urma startării platformei ţintă (sau după o operaţie de RESET), execuţia procesorului este redirectată către zona de start a codului secvenţei BOOT din memoria EPROM a sistemului.

   Odată activat, task-ul BOOT încarcă în memoria sistemului (RAM) structurile de date şi codul celorlalte componente ale nucleului, după care lansează iniţializarea sistemului (apelează task-ul SYSINIT).

2. În faza de iniţializare a sistemului, task-ul lejer SYSINIT setează valorile parametrilor de operare ai nucleului, variabilele globale, parametrii şi variabilele celorlalte componente ale nucleului.

   Lansarea task-ului SYSINIT corespunde practic cu activarea contextului lejer de operare al HARETICK. După iniţializarea parametrilor sistemului, SYSINIT configurează şi porneşte ceasul de timp-real (RTC) care, prin intermediul întreruperii de timer apelează executivul TR al HARETICK, HDIS. Prin aceasta, contextul strict de operare al sistemului este, la rândul lui, activat.

3. Odată activat contextul strict, execuţia SYSINIT este întreruptă de către HDIS, care identifică din Tabela de Dispatch, ModX-ul care urmează a fi lansat în execuţie. În această fază de operare a sistemului, singurul ModX prezent în tabelă este planificatorul pentru contextul strict, HSCD. Prin urmare, HSCD este apelat de către HDIS.

   Înainte de a ceda procesorul către HSCD, executivul HDIS apelează subrutina de raportare a evenimentelor sistem, TiLT, pentru a marca lansarea în execuţie a lui HSCD.

4. Planificatorul de ModX-uri, HSCD, va aplica algoritmii de planificare non-preemptivă special concepuţi, asupra setului de ModX-uri existente în momentul curent în cadrul sistemului (referiţi în PDT – tabela cu descriptorii de proces). Planificarea ModX-urilor durează până în momentul în care HSCD a completat toate înregistrările din Tabela de Dispatch cu indentificatorii şi momentele de start corespunzătoare ciclului curent de planificare.

   Ca observaţie, menţionăm ca HSCD va modifica şi înregistrarea proprie din Tabela de Dispatch, setând pentru momentul de start, noua valoare la care este planificat HSCD după execuţia ultimului ModX în cadrul ciclului de planificare.

5. Contextul de execuţie strictă se va continua după terminarea execuţiei planificatorului HSCD, prin lansarea celorlalte ModX-uri din Tabela de Dispatch la momentele de timp corespunzătoare.

   Printre aceste ModX-uri, aparţinând configuraţiei iniţiale de operare a nucleului HARETICK, se regăsesc ModX-urile componentei DATALINK, ce asigură comunicaţia de comenzi şi date cu un calculator gazdă.

   În Figura 8-1 sunt reprezentate două bucle de execuţie pentru contextul strict, subliniind faptul că fiecare ModX este lansat în execuţie de către HDIS şi că planificarea acestora se face ciclic, pe baza Tabelei de Dispatch, de către HSCD care se execută odată la fiecare ciclu.

6. În intervalele de timp neocupate de execuţia contextului strict, se vor executa task-urile contextului lejer. Prima dată când controlul execuţiei revine contextului lejer, se continuă execuţia task-ului SYSINIT care termină iniţializările celorlalte task-uri (aparţinând contextului lejer) şi lansează planificatorul task-urilor lejere, SSCD.

7. SSCD are rolul de a planifica, în intervalele de timp neocupate de execuţia contextului strict al HARETICK, task-urile lejere ale sistemului şi ale aplicaţiei (după ce se încarcă o anumită aplicaţie în sistem). În această fază a scenariului, task-urile lejere considerate de algoritmul de planificare preemptivă al SSCD sunt: task-urile lejere ale DATALINK, task-ul MONITOR, LOADER şi STATREPO. Acestea sunt planificate şi executate după un algoritm de planificare de tip "time-sharing", preemptiv, pe bază de priorităţi.

   În Figura 8-1 este reprezentată bucla de execuţie corespunzătoare contextului lejer al HARETICK, definită de task-ul de planificare SSCD.

9 Reprezentarea şi gestionarea timpului şi a ModX-urilor în cadrul sistemului

9.1 Timpul ca parametru esenţial de operare al HARETICK

Timpul, ca şi coordonată esenţială a sistemelor TR stricte, trebuie să fie implicat în toate fazele de dezvoltare a acestora: specificare şi verificare formală, programare, analiză, planificare şi execuţie, şi implementare (aşa cum se subliniază în secţiunea anterioară). În consecinţă, întreaga concepţie şi implementare a nucleului HARETICK ţine cont de acest obiectiv principal.

În cele ce urmează vom discuta aspectele legate de reprezentarea şi gestionarea timpului în cadrul nucleului TR HARETICK pentru sisteme de control încorporat şi DSP. Acolo unde se impun particularizări, se vor exemplifica abordările din cadrul implementării HARETICK pe platforma Motorola DSP56307EVM [Motorola 95, Motorola 98, Motorola 99, Motorola 00].

Frecvenţa tactului de core a procesorului DSP56307 a fost setată la valoarea:

$$CoreCLK = 32 \text{ MHz}.$$

În cazul acestei arhitecturi, ciclul procesor ce defineşte execuţia instrucţiunilor este egal cu o perioadă de tact, rezultând astfel granularitatea de timp minimă ca fiind egală cu durata ciclului procesor (vezi Capitolul 4):

$$\Delta t_{CLK} = 31{,}25 \text{ ns}.$$

Pe de altă parte, se va utiliza un ceas de timp-real (RTC) ce aplică tactului intern al DSP un factor de divizare (prescalare) egal cu 4. Rezultă pentru unitatea de timp sistem (unitatea de timp măsurată de RTC), valoarea:

$$\Delta t_{RTC} = 4 \cdot \Delta t_{CLK} = 125 \text{ ns}.$$

Astfel, HARETICK poate determina evenimentele din cadrul sistemului cu o frecvenţă maximă de 8 MHz = *CoreCLK* / 4, echivalentă cu 4 cicli procesor.

Observaţie

Frecvenţa de lucru a procesorului şi unitatea de timp sistem au fost stabilite la valorile de mai sus în implementarea HARETICK din considerente practice.

Astfel, pentru frecvenţa de lucru a DSP s-a căutat o valoare comodă din punctul de vedere al calculelor legate de timp. Iniţial a fost încercată programarea blocului PLL (care generează practic frecvenţa de lucru internă a DSP pe baza unui oscilator extern, de frecvenţă F_{ext} = 12.288 MHz), pentru valoarea *CoreCLK* = 80 MHz. Arhitectura internă şi parametrii de operare ai PLL nu au permis însă o astfel de setare.

Unitatea de timp sistem Δt_{RTC} a fost stabilită la 4 cicli instrucţiune din considerente legate de relaxarea preciziei cu care se calculează timpii de execuţie ai ModX-urilor, oferindu-se astfel un interval de toleranţă destinat acoperii eventualelor erori şi imprecizii ce derivă din calculul acestor timpi.

9.2 Structurile destinate reprezentării şi gestionării timpului în HARETICK

Timpul sistem este reprezentat şi gestionat în cadrul HARETICK utilizând un set de structuri şi mecanisme hardware şi software, ce vor fi descrise pe scurt în continuare.

9.2.1 Timpul absolut sistem

Nucleul HARETICK implementează o structură pentru gestionarea timpului absolut, utilizând timpul sistem intern şi setări ale operatorului de sistem, provenite din exterior (de la interfaţa INVERTA), prin intermediul unor comenzi specifice.

Timpul absolut sistem este stocat într-o variabilă sistem, `Sys_AbsTime`, de dimensiune egală cu 3 cuvinte procesor:

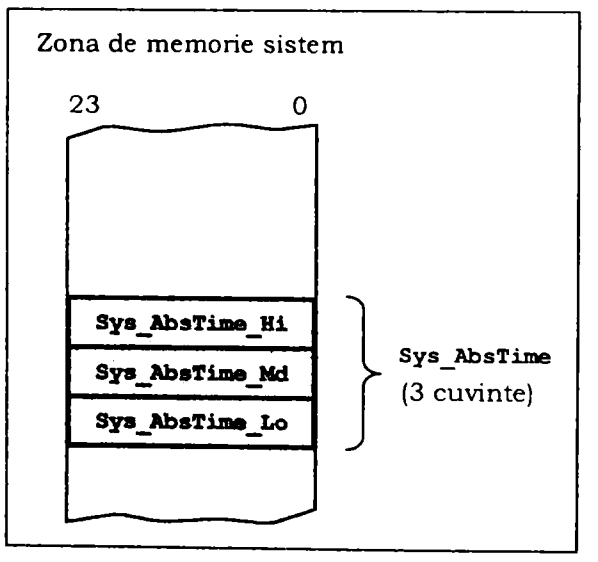

Figura 9-1. Reprezentarea timpului absolut sistem

Ţinând cont că Motorola DSP56307 este o arhitectură pe 24 biţi, timpul absolut sistem este exprimat pe 72 biţi. În aceste condiţii, şi considerând setările pentru unitatea de timp sistem (Δt_{RTC} = 125 ns), timpul absolut pe care îl poate reprezenta şi gestiona sistemul HARETICK se întinde pe o durată maximă:

$$\max \{\texttt{Sys\_AbsTime}\} = 18.718.157{,}355 \text{ ani.}$$

Parametrul sistem `Sys_AbsTime` poate fi setat (actualizat) de către operator prin intermediul unor comenzi specifice, lansate din mediul INVERTA, la valoarea timpului curent sau după necesităţi. În timpul operării sistemului HARETICK,

`Sys_AbsTime` este actualizată în mod constant, ciclic, de către ModX-ul HSCD (planificatorul contextului strict de execuţie). Actualizarea timpului absolut sistem se face în cadrul fiecărei execuţii a HSCD, cu valoarea acumulată a intervalului de timp corespunzător ultimului ciclu de planificare.

9.2.2 Ceasul timp-real

Ceasul timp-real (RTC) este practic elementul activ de gestionare a timpului în sistemul HARETICK. RTC are două roluri esenţiale:

(a) Contor de timp, utilizat pentru referiri ale componentelor sistemului la instanţa curentă de timp sistem. Orice task, fie că aparţine nucleului HARETICK, fie aplicaţiei ce rulează la un moment dat în sistem, poate citi valoarea curentă a contorului RTC, având astfel la dispoziţie o bază de timp real. Fiind vorba de un sistem TR (strict) destinat aplicaţiilor critice, accesul oricărui task la baza de timp a sistemului reprezintă o facilitate extrem de utilă şi necesară.

(b) Singura sursă (predictibilă) de întreruperi.

Aşa cum a fost subliniat în cadrul secţiunilor anterioare, abordarea pe care o propunem şi o adoptăm în ceea ce priveşte dezvoltarea sistemelor TR stricte presupune restricţionarea pe cât posibil a mecanismului de întreruperi.

În implementarea curentă a nucleului HARETICK, ceasul timp-real reprezintă singura sursă de întreruperi din cadrul sistemului. Timerele RTC sunt programate de către executivul TR al nucleului, HDIS, pentru a genera câte o întrerupere în momentul corespunzător lansării în execuţie a fiecărui ModX, conform planificării calculate de HSCD şi înscrise în Tabela de Dispatch.

Mai mult, RTC reprezintă o sursă predictibilă de întreruperi, în sensul că se poate determina oricând, şi de către orice task, momentul apariţiei următoarei întreruperi. Acest lucru se poate realiza în două moduri: (i) prin citirea valorii înscrisă la un moment dat în regiştrii de comparare ai RTC, sau (ii) prin citirea înregistrării din Tabela de Dispatch, corespunzătoare ModX-ului ce urmează a fi executat după cel curent. Înregistrarea respectivă conţine informaţia de timp referitoare la momentul începerii execuţiei ModX-ului, corespunzând următoarei întreruperi RTC.

Pentru a uşura operarea şi utilizarea ceasului timp-real, este de dorit o variantă bazată pe un contor de timp cu capacitate cât mai mare. Implementarea curentă a sistemului HARETICK se bazează pe arhitectura procesorului Motorola DSP56307 [Motorola 98], care prevede existenţa a 3 timere interne, pe 24 biţi fiecare. Astfel, RTC poate fi implementat cu ajutorul a două timere cascadate, capacitatea totală de contorizare a acestuia fiind prin urmare de 48 biţi.

Figura 9-2 prezintă arhitectura RTC din HARETICK, bazată pe cascadarea a două timere pe 24 biţi fiecare, `Timer0` şi `Timer1`, cu numărare crescătoare. În aceste condiţii, şi considerând setările pentru unitatea de timp sistem (Δt_{RTC} = 125 ns), timpul sistem pe care îl poate reprezenta şi gestiona RTC se întinde pe o durată maximă:

max {RTC} = 407,226 zile,

iar capacitatea maximă a contorului Timer0 al RTC este de:

max {Timer0} ≅ 2 sec.

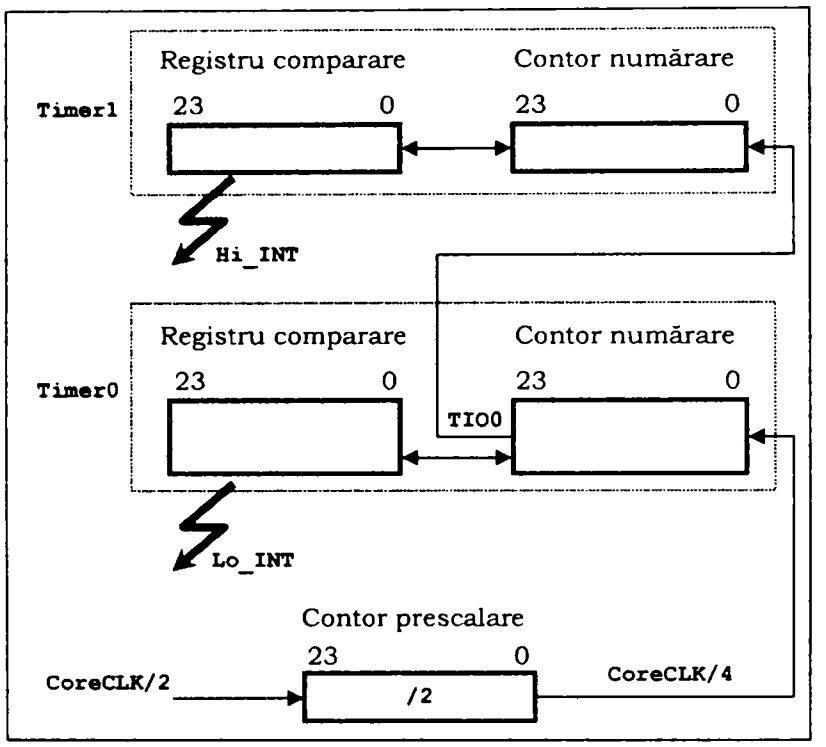

Figura 9-2. Ceasul timp-real (RTC) în HARETICK (varianta cu timere cascadate)

Cascadarea celor două timere ale DSP56307 pentru obținerea unei capacități suficiente de contorizare a timpului real complică mecanismul de întreruperi al HARETICK, în sensul că introduce două întreruperi RTC în loc de una: Lo_INT, şi Hi_INT. Întreruperile se activează atunci când valoarea contorului de numărare ajunge la valoarea din registrul de comparare corespunzător, pentru fiecare timer în parte. Mecanismul de întreruperi RTC va fi detaliat în cadrul prezentării executivului TR al HARETICK, HDIS, acesta fiind ModX-ul care lucrează în mod direct cu ceasul timp-real (Capitolul 10).

Pentru situaţiile în care timerele procesorului ţintă nu pot fi cascadate sau nu sunt disponibile mai multe timere, o a doua variantă de implementare a RTC se poate lua în considerare (vezi Figura 9-3). Cel mai puţin semnificativ cuvânt (LSW) al RTC este reprezentat de contorul unui timer al procesorului, iar cel mai semnificativ cuvânt (MSW) este reprezentat printr-o variabilă dedicată, Sys_RTC_MSW, alocată în memoria de date sistem. Contorul Timer0 este configurat pentru numărare în sens crescător, intrarea de tact fiind dată direct de tactul intern al procesorului (CoreCLK), căruia i se poate aplica un factor de divizare. În momentul în care are loc o depăşire a capacităţii contorului Timer0, se generează o întrerupere, RTC_OVR, a cărei subrutină de tratare incrementează valoarea variabilei sistem Sys_RTC_MSW. Întreruperea RTC a HARETICK, utilizată pentru apelarea executivului este RTC_INT, care se activează în condiţiile în care valoarea variabilei Sys_RTC_MSW a ajuns egală cu partea cea mai

semnificativă a instanței de start pentru următoarea planificare şi, în acelaşi timp, contorul `Timer0` a ajuns egal cu partea cea mai puțin semnificativă a momentului următoarei planificări. `RTC_INT` este astfel, o întrerupere de comparare generată de `Timer0`. Şi în această variantă de implementare, apare problema tratării unei întreruperi suplimentare, anume `RTC_OVR`.

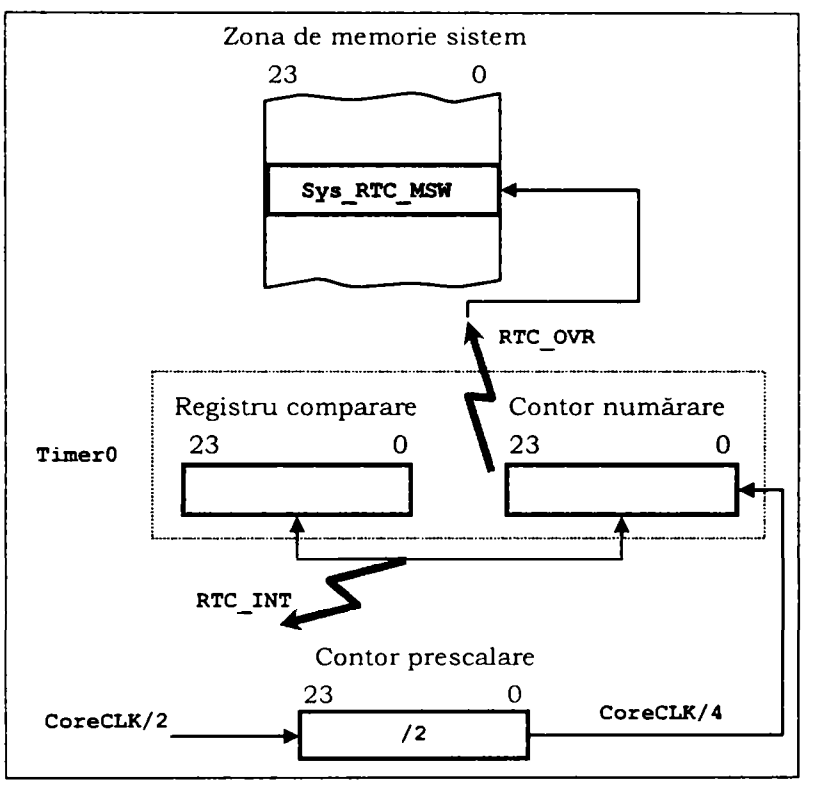

Figura 9-3. Ceasul timp-real (RTC) în varianta cu un singur timer

Gestionarea timpului în cadrul RTC începe în momentul activării contextului strict de execuție al HARETICK, operație inițiată de către task-ul SYSINIT. În acel moment valoarea timpului real sistem este:

$$t_{RTC} = t_0 = 0$$

Pe lângă executivul TR al nucleului (HDIS), ceasul timp-real mai este utilizat de către planificatorul contextului strict, HSCD, pentru calcularea şi exprimarea timpilor de start ai ModX-urilor planificate. HSCD nu foloseşte însă în mod direct RTC şi nu-i modifică valoarea, ci utilizează structuri proprii pentru exprimarea timpului de planificare. Acestea vor fi descrise în paragraful următor.

9.2.3 Timpul de planificare

Planificatorul contextului strict de execuție al HARETICK, anume ModX-ul HSCD, gestionează, la rândul său, timpul, dispunând de structuri şi mecanisme proprii pentru aceasta.

Aşa cum a fost detaliat în secțiunea anterioară, algoritmii de planificare propuşi calculează planificarea ModX-urilor pe un interval de timp de forma:

$$T_{SCHED} = [0, T_{LCM}),$$

unde T_{LCM} reprezintă cel mai mic multiplu comun al perioadelor ModX-urilor din setul supus planificării.

În consecinţă, ModX-ul de planificare HSCD, utilizează aşa-numitul *timp absolut de planificare*, t_{SCHED}, care este definit pe intervalul $[0, T_{LCM})$. Dat fiind faptul că lungimea intervalului de planificare (valoarea lui T_{LCM}) variază puternic cu perioadele şi numărul ModX-urilor, putând ajunge la valori foarte mari, în implementarea HARETICK se utilizează o variabilă de capacitate suficientă (vezi Figura 9-4).

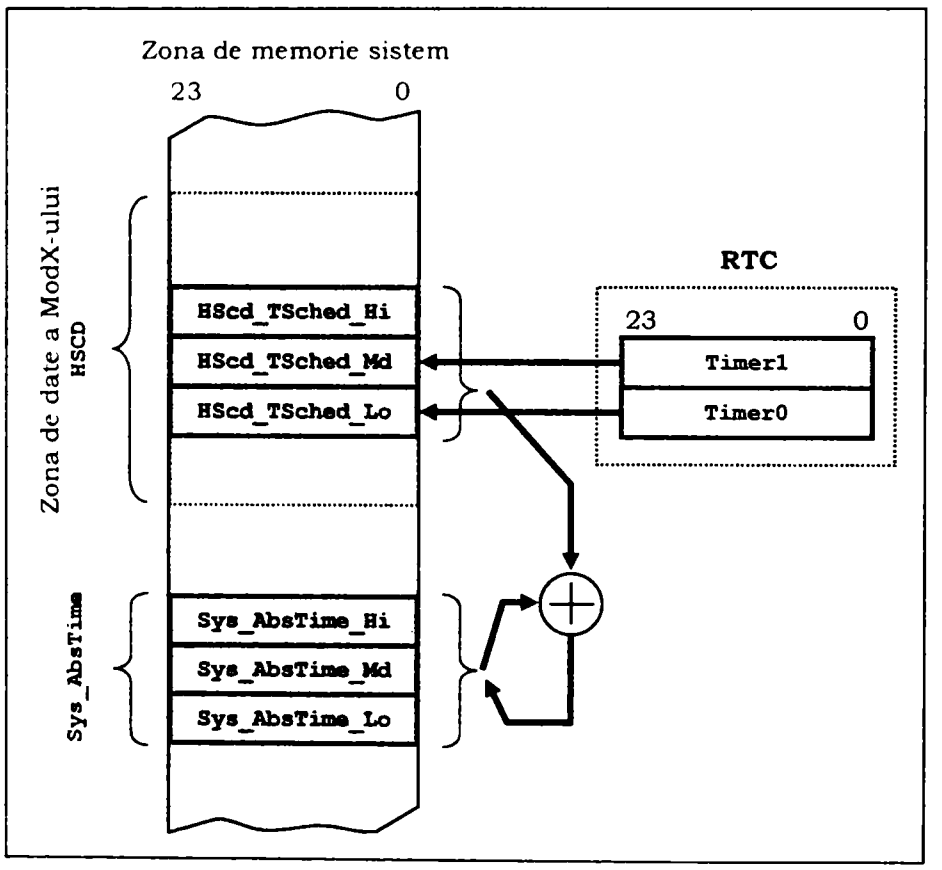

Figura 9-4. Structurile şi mecanismele de gestionare a timpului absolut de planificare

Cum rezultă şi din figură, timpul absolut de planificare (sau, simplu – *timpul de planificare*) este exprimat pe 3 cuvinte a câte 24 biţi, rezultând o capacitate totală de 72 biţi (identică cu cea corespunzătoare timpului absolut sistem). Figura 9-4 indică şi faptul că timpul de planificare este sincronizat cu cele două timere ale RTC.

Timpul de planificare calculat de către algoritmul de planificare implementat de HSCD este apoi utilizat pentru completarea înregistrării corespunzătoare din Tabela de Dispatch. În tabelă, HDIS va înscrie, pe lângă identificatorul ModX-ului planificat, şi valoarea timpului real sistem la care va fi lansat în execuţie ModX-ul respectiv de către HDIS. Astfel, componenta temporală a unei înregistrări din Tabela de Dispatch va avea valoarea `HScd_TSched_Md:HScd_TSched_Lo`.

9.3 Mecanismele de sincronizare a structurilor de gestionare a timpului în HARETICK

Relaţiile dintre cele trei tipuri de timpi utilizaţi în cadrul sistemului HARETICK, adică timpul absolut sistem, timpul real sistem (RTC) şi timpul de planificare, sunt bine definite:

- Există o singură structură activă dc generare şi măsurare a timpului în cadrul sistemului: RTC, care gestionează timpul real sistem – baza de timp a sistemului.
- Timpul real sistem (t_{RTC}) este generat (modificat) doar de timerele procesorului şi este referit de componentele de bază ale nucleului HARETICK: HDIS, HSCD, etc. El poate fi de asemenea citit de către alte task-uri (stricte sau lejere) aparţinând nucleului sau aplicaţiei curente.
- Iniţializarea RTC este efectuată de către task-ul SYSINIT, o singură dată, la pornirea sistemului
- Timpul absolut sistem poate fi setat şi referit la nivelul operatorului, prin intermediul unor comenzi specifice, şi nu este utilizat de nucleu la operaţii de sincronizare a execuţiilor sau pentru planificarea ModX-urilor.
- ModX-ul de planificare a contextului strict de execuţie, HSCD, utilizează timpul de planificare (t_{SCHED}), sincronizându-l cu timpul real sistem (RTC) la fiecare execuţie. La începutul fiecărui ciclu de planificare, cu ocazia lansării sale în execuţie, HSCD va efectua următoarele acţiuni:

  - Verifică relaţia de corespondenţă dintre timpul de planificare şi RTC, şi în caz că e necesar, efectuează sincronizarea:

$$\begin{cases} \texttt{HScd\_TSched\_Md} \leftarrow \texttt{Timer1} \\ \texttt{HScd\_TSched\_Lo} \leftarrow \texttt{Timer0} \end{cases}$$

  - Sincronizează timpul absolut sistem cu timpul de planificare:

$$\begin{cases} \texttt{Sys\_AbsTime\_Hi} \leftarrow \texttt{Sys\_AbsTime\_Hi} + \texttt{HScd\_TSched\_Hi} \\ \texttt{Sys\_AbsTime\_Md} \leftarrow \texttt{Sys\_AbsTime\_Md} + \texttt{HScd\_TSched\_Md} \\ \texttt{Sys\_AbsTime\_Lo} \leftarrow \texttt{Sys\_AbsTime\_Lo} + \texttt{HScd\_TSched\_Lo} \end{cases}$$

  - Aplică algoritmul de planificare non-preemptivă a setului de ModX-uri existent în sistem, pentru completarea înregistrărilor din Tabela de Dispatch cu planificările ciclului curent. Astfel, dacă la momentul curent al planificării, $t_{SCHED.i}$, a fost selectat pentru execuţie ModX-ul M_j, înregistrarea curentă din Tabela de Dispatch va conţine următoarele elemente (MSW semnifică cel mai semnificativ cuvânt, iar LSW – cel mai puţin semnificativ):

$$\begin{cases} 1.\ \text{Identificatorul ModX - ului}: M_j \\ 2.\ \text{MSW al momentului planificarii } \left(t_{st}^{M_j}\right): \left(t_{SCHED,i}\right).\texttt{HScd\_TSched\_Md} \\ 3.\ \text{LSW al momentului planificarii } \left(t_{st}^{M_j}\right): \left(t_{SCHED,i}\right).\texttt{HScd\_TSched\_Lo} \end{cases}$$

    - După completarea Tabelei de Dispatch, HSCD configurează înregistrarea specială din tabelă ce face referire la acesta, cu timpul de start al următoarei sale execuţii – specificând începutul unui nou ciclu de planificare.

- Momentele de start ale execuţiei ModX-urilor planificate în cadrul unui ciclu de planificare vor fi referite pe rând, din Tabela de Dispatch, de către executivul nucleului, HDIS. Înaintea lansării ModX-ului curent, HDIS programează registrele de comparare ale RTC cu cele două cuvinte (MSW, respectiv LSW) ale timpului de start corespunzător ModX-ului ce urmează celui curent.

Operaţiile referitoare la timp, desfăşurate de componentele HARETICK, vor fi detaliate în cadrul secţiunilor ce descriu componentele respective în cadrul lucrării.

9.4 Reprezentarea ModX-urilor în cadrul HARETICK

Task-urile cu specificaţii temporale stricte ale unui sistem sau aplicaţii timp-real trebuie tratate cu atenţie specială. Modelul propus pentru astfel de task-uri în cadrul lucrării de faţă este *ModX*-ul (*Modul eXecutabil*), ale cărui caracteristici au fost studiate în secţiunea precedentă: un task TR periodic, modular, cu specificaţii complete şi stricte de comportare în timp şi cu planificare şi execuţie în context non-preemptiv.

Nucleul de operare HARETICK implementează un set de structuri şi mecanisme destinate reprezentării, planificării şi execuţiei ModX-urilor. Dat fiind tipul sistemului considerat – nucleu de operare TR strict pentru platforme bazate pe DSP sau platforme de control digital încorporat (embedded systems), reprezentarea ModX-urilor se face pe baza unei arhitecturi pe cât posibil statice.

ModX-urile gestionate la un moment dat de nucleul HARETICK se împart în două categorii: ModX-uri sistem şi ModX-uri aplicaţie. HARETICK utilizează structuri distincte pentru cele două categorii de ModX-uri, deşi din punctul de vedere al tipului de sistem implementat (sistem încorporat), această diferenţiere nu este obligatorie (de exemplu, din acest motiv nu a fost prevăzut nici un mecanism de control al accesului task-urilor la memoria sistemului, rezolvarea acestei chestiuni fiind responsabilitatea programatorului de sistem/aplicaţie).

ModX-urile sistem sunt încărcate de către task-ul BOOT la pornirea platformei, alocându-se pentru ele un set de resurse distincte (de exemplu, în cazul memoriei, există o zonă dedicată ModX-urilor sistem – zona sistem). ModX-urile sistem vor fi discutate separat în cadrul lucrării de faţă, făcând parte integrantă din nucleul HARETICK.

În continuare vom discuta cazul general al unui ModX ce aparţine unei aplicaţii oarecare, încărcate pe platforma ţintă ce operează sub nucleul HARETICK.

O primă structură implicată în gestionarea ModX-urilor aplicaţiei este tabela ce conţine graful direcţionat aciclic de program al aplicaţiei (*PDAGT – Program Directed Acyclic Graph Table*). Fiecare înregistrare a tabelei constituie o reprezentare a unui nod al grafului (adică a unui ModX al aplicaţiei):

- identificatorul ModX-ului: `PID`;
- numărul de nivel în cadrul grafului;
- numărul de părinţi ai nodului;
- referinţă către părintele (părinţii) nodului;
- numărul de fii ai nodului;
- referinţă către fiul (fiii) nodului.

Până în momentul de faţă a fost studiată reprezentarea aplicaţiilor ce rezultă în grafuri de program de tip arbore (fiecare nod are un singur părinte şi unul sau mai mulţi fii). Ca preocupare de viitor, avem în vedere conceperea de structuri (statice) care să poată reprezenta grafuri de program complexe (un nod poate avea mai mulţi părinţi şi mai muţi fii).

Tabela PDAGT este creată de către task-ul LOADER al nucleului HARETICK, în faza de încărcare a aplicaţiei în cadrul sistemului ţintă, prin intermediul interfeţei de comunicaţie cu mediul INVERTA, implementată de DATALINK. Translatarea structurii aplicaţiei, dintr-o reprezentare dinamică de tip graf, într-una statică de tip tabelă este posibilă datorită operaţiilor de analiză a aplicaţiei dezvoltate în mediul INVERTA, pe calculatorul gazdă. Astfel, printre informaţiile suplimentare furnizate de INVERTA la încărcarea aplicaţiei în sistemul HARETICK, se găseşte şi un set de parametri ce definesc cantitativ graful aplicaţiei (numărul total de noduri – ModX-uri, numărul de nivele ale grafului, etc.).

9.4.1 Reprezentarea ModX-urilor în memorie

Tot în cadrul operaţiilor de încărcare a aplicaţiei în sistem, task-ul LOADER alocă şi completează, pentru fiecare ModX, un set de structuri de memorie, într-o zonă dedicată: zona aplicaţie.

Pentru fiecare ModX, sunt prevăzute următoarele structuri de memorie:

- Zona de cod (text), în care se încarcă secvenţa de cod compilat a ModX-ului;
- Zona parametrilor de ieşire, ce conţine valorile fiecărui parametru de ieşire al ModX-ului.

De asemenea, pentru întreaga aplicaţie, este alocată o zonă de memorie pentru variabilele globale, şi o zonă de memorie comună pentru parametrii de intrare şi variabilele locale ale ModX-urilor, denumită *Heap*.

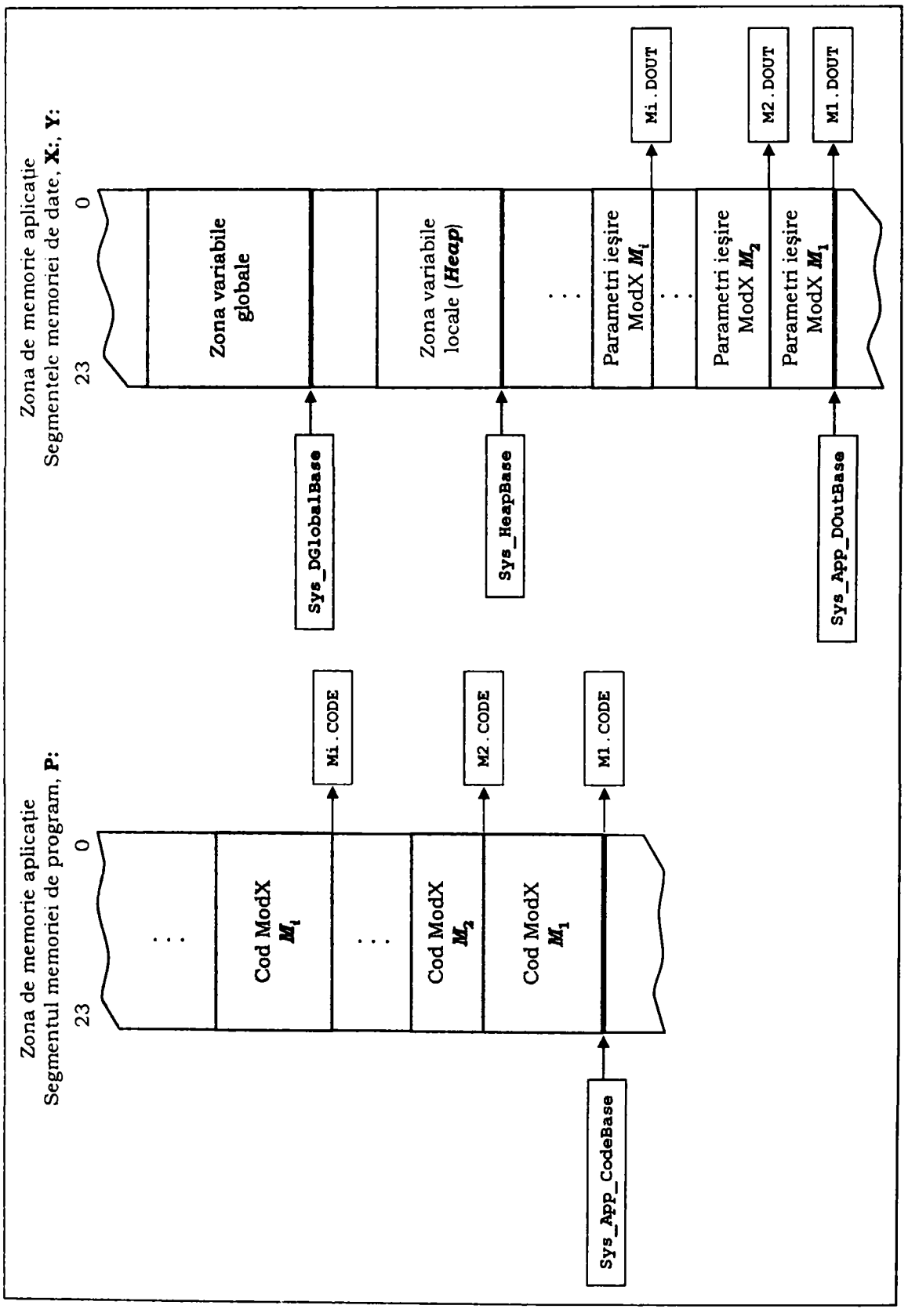

Figura 9-5. Reprezentarea ModX-urilor în memorie

La fel ca în cazul tabelei PDAGT, toate structurile utilizate de sistem pentru reprezentarea în memorie a ModX-urilor sunt definite şi alocate static, pe baza informaţiilor suplimentare legate de dimensiunile maxime ale fiecărei structuri, primite de HARETICK de la mediul de dezvoltare INVERTA, la încărcarea aplicaţiei în sistem. De exemplu, pentru zona *Heap*, LOADER-ul va aloca o zonă de memorie de dimensiune cel puţin egală cu cea maximă necesară oricărui ModX din cele ale aplicaţiei pentru lucrul cu variabilele sale locale (şi parametrii de intrare).

Figura 9-5 prezintă structurile necesare nucleului HARETICK pentru reprezentarea în memorie a ModX-urilor aplicaţiei. Codul fiecărui ModX este înscris în segmentul de memorie program al procesorului DSP56307 (memoria **P:**), începând de la o adresă de bază, `Sys_App_CodeBase`, care constituie un parametru sistem. În acelaşi timp, LOADER-ul reţine adresele de start ale zonei de cod pentru fiecare ModX (`Mi.CODE`), pentru a le înscrie în celelalte tabele care servesc nucleului la gestionarea ModX-urilor, şi care vor fi discutate în paragrafele următoare.

Zona parametrilor de ieşire ai fiecărui ModX este alocată în segmentul de memorie de date a DSP56307 (memoria **X:** sau **Y:**), începând de la o adresă de bază, `Sys_App_DOutBase`, care este, de asemenea, un parametru sistem.

Zona *Heap* este alocată de către LOADER, pe considerentele de dimensiune prezentate mai sus, începând de la adresa `Sys_HeapBase`, şi, în manieră similară se procedează cu zona variabilelor globale ale aplicaţiei, folosind ca adresă de bază, `Sys_DGlobalBase`.

9.4.2 Tabela de simboluri

În aceeaşi fază de încărcare a aplicaţiei în cadrul sistemului HARETICK, task-ul LOADER creează tabela de simboluri, în care este înscrisă corespondenţa dintre fiecare parametru sau variabilă a ModX-urilor şi locaţia de memorie corespunzătoare.

Astfel, de exemplu, pentru primul parametru de ieşire al unui ModX oarecare M_i (adică pentru parametrul alocat la startul zonei `Mi.DOUT`), LOADER-ul va trece în tabela de simboluri numele parametrului şi adresa corespunzătoare valorii `Mi.DOUT`.

Pentru parametrii de intrare sau variabilele interne ale unui ModX, tabela de simboluri va conţine numele variabilei şi deplasamentul stabilit de către LOADER în cadrul zonei *Heap* pentru variabila respectivă. Va rezulta astfel o suprapunere a adreselor din tabela de simboluri pentru aceste variabile, prin maparea lor într-o aceeaşi zonă fizică de memorie.

9.4.3 Tabela descriptorilor de procese

Tabela descriptorilor de procese (*PDT – Process Descriptor Table*) este creată şi completată tot de către LOADER, în faza de încărcare a aplicaţiei în sistem. PDT conţine identificarea fiecărui ModX, împreună cu caracteristicile sale (atribute, parametri temporali, etc.).

Figura 9-6 prezintă structura PDT şi a unei înregistrări, care are o dimensiune de 13 cuvinte (de 24 biţi în cazul arhitecturii DSP56307), şi corespunde unui ModX oarecare. Semnificaţia elementelor înregistrării PDT este prezentată în Tabelul 9-1.

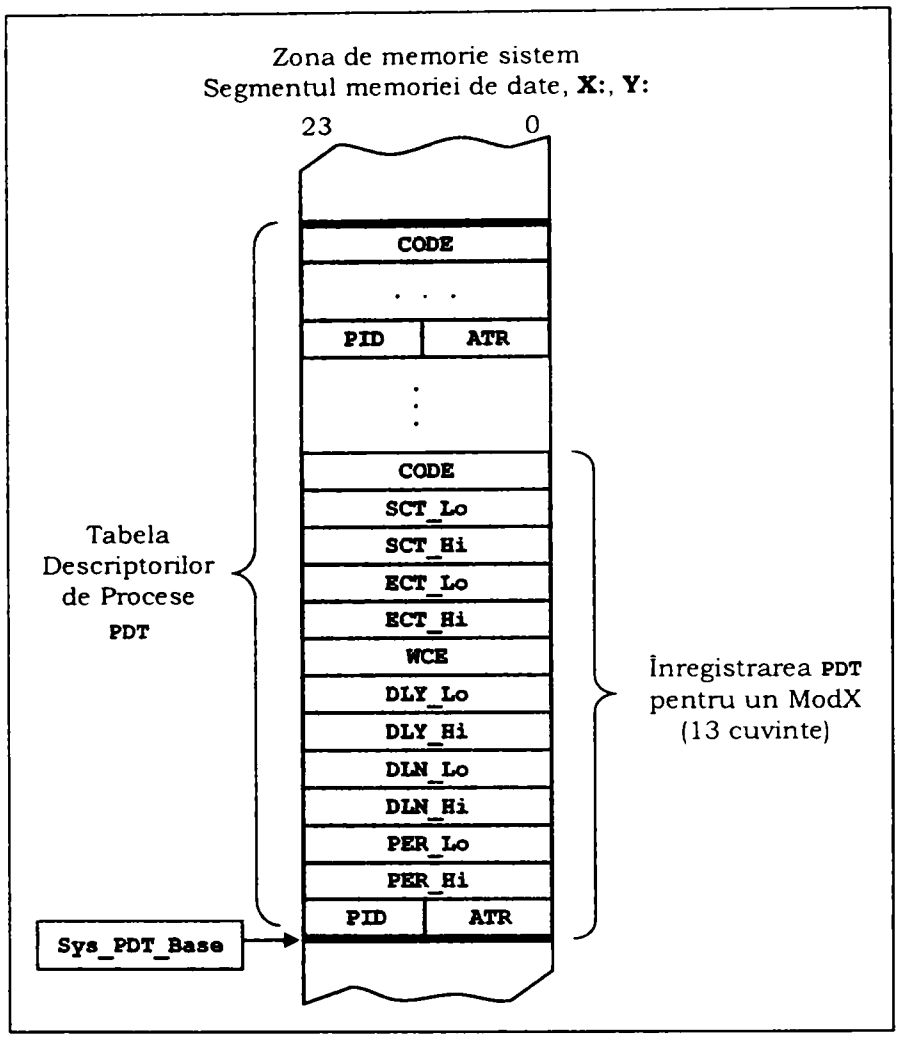

Figura 9-6. Tabela Descriptorilor de Procese (PDT)

Tabelul 9-1. Semnificaţia elementelor unei înregistrări din PDT (vezi şi Capitolul 4)

Offset	Denumire	Semnificaţie
0	PID (12 biţi)	Identificatorul de proces, pentru ModX-ul M_i.
	ATR (12 biţi)	Atribute suplimentare ModX (e.g. "FIXED", etc.).
1:2	PER_Hi:PER_Lo	Perioada ModX-ului (period), $T_{pr}^{M_i}$.
3:4	DLN_Hi:DLN_Lo	Intervalul limită (deadline), $T_{dl}^{M_i}$.
5:6	DLY_Hi:DLY_Lo	Intervalul de întârziere (delay), $T_{dy}^{M_i}$.
7	WCE	Durata maximă de execuţie (execution time), $T_{ex}^{M_i}$.
8:9	ECT_Hi:ECT_Lo	Contorul de execuţii (execution count), N^{M_i}.
10:11	SCT_Hi:SCT_Lo	Contorul de planificări (scheduling count) – utilizat de algoritmul de planificare al HSCD.
12	CODE	Adresa de start a zonei de cod a ModX-ului M_i.

Majoritatea parametrilor din PDT corespunzători unui ModX sunt utilizaţi de către HSCD, în cadrul algoritmului de planificare non-preemptivă a contextului strict de execuţie. PDT este referită de asemenea de către executivul TR al HARETICK, HDIS, pentru identificarea ModX-ului ce urmează a fi lansat în execuţie (PID), pentru încărcarea adresei de start a zonei de cod a ModX-ului (CODE) şi pentru accesarea (în citire-scriere) a contorului de execuţii (ECT).

Din punct de vedere particular, al implementării HARETICK pe platforma Motorola DSP56307, structura PDT permite următoarele valori maximale pentru parametrii sistem şi ai ModX-urilor:

- Dimensiune PID: 12 biţi, rezultând un total de 4 K (4096) ModX-uri ce pot fi gestionate de către HARETICK;
- Dimensiune totală PDT: 100 înregistrări, rezultând un total de 100 ModX-uri ce pot fi gestionate de către sistem;
- Perioada unui ModX, PER, e exprimată pe 2 cuvinte (48 biţi), rezultând un interval maxim de timp de 407,226 zile;
- Acelaşi interval maxim de timp poate fi exprimat pentru intervalul limită al planificării (DLN) şi pentru intervalul de întârziere (DLY) ale ModX-ului: 407,226 zile;
- Durata maximă de execuţie a ModX-ului (WCE) poate ajunge la 2 sec (aproximativ), parametrul fiind exprimat pe un cuvânt procesor (24 biţi);
- Contorul de execuţii al ModX-ului (ECT) are o dimensiune de 2 cuvinte (48 biţi), rezultând un număr maxim de peste $281 \cdot 10^{12}$ execuţii ale ModX-ului ce pot fi gestionate de sistem;
- Contorul de planificări (SCT) poate exprima, la rândul lui, peste $281 \cdot 10^{12}$ planificări ale ModX-ului.

Atributul "FIXED" specificat în Tabelul 9-1 ca exemplificare pentru parametrul ATR al ModX-urilor se referă la cerinţa de execuţie fixă a unui ModX în cadrul perioadelor acestuia (cu alte cuvinte, faţă de momentul de început al fiecărei perioade, ModX-ul respectiv se va executa cu o aceeaşi întârziere). Această problemă a fost tratată din punct de vedere teoretic şi exemplificată, în Capitolul 5.

9.5 Operarea de principiu a ModX-urilor în cadrul HARETICK

Ca linii generale, pentru satisfacerea cerinţelor de predictibilitate impuse de aplicaţiile critice, HARETICK utilizează structuri statice, limitate în timp şi spaţiu. Pentru fiecare aplicaţie încărcată în sistem se vor cunoaşte parametrii maximali de operare ai acesteia: timpi, resurse sistem, memorie ocupată, etc.

În cele ce urmează, vom exemplifica operarea sistemului pentru cazul a două ModX-uri, M_i şi M_j, care aparţin unei aplicaţii oarecare, încărcată pe platforma sistemului. Presupunem că M_i şi M_j se află într-o relaţie de precedenţă de forma: $M_j \rightarrow M_i$, precedenţă rezultată ca urmare a dependenţei de control şi de date a aplicaţiei. Figura 9-7 ilustrează acest scenariu, reprezentând cele două noduri (ModX-uri) ale grafului aplicaţiei, împreună cu dependenţele de program corespunzătoare. Fiecare nod al grafului va avea specificaţii complete: numele (M_j,

M_i), parametrii de intrare (paranteza superioară denumirilor ModX-urilor), parametrii de intrare (paranteza inferioară), parametrii de comportare temporală, etc. De exemplu, ModX-ul M_i are 3 parametri de intrare (`param1`, `param2` şi `param3`) şi un parametru de ieşire (`param3`). Primele două intrări ale lui M_i provin de la parametrii de ieşire ai lui M_j (de unde rezultă şi dependenţa de date dintre cele două), iar `param3` reprezintă în acelaşi timp, parametru de intrare şi de ieşire, pentru M_i.

Posibilitatea ca un ModX oarecare să aibă parametri care constituie în acelaşi timp intrare şi ieşire, derivă din modelul conceput pentru task-urile TR stricte: prin acest mecanism ele îşi pot transmite informaţii de stare, de la o execuţie la alta (adică de la o perioadă la alta).

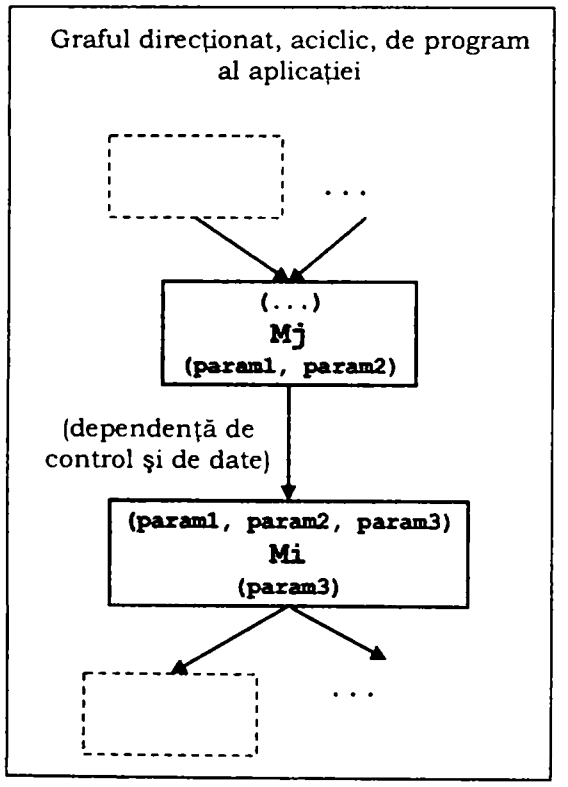

Figura 9-7. Exemplificare cu porţiunea de graf al aplicaţiei ce conţine cele două ModX-uri, M_j şi M_i

Presupunem de asemenea, în cadrul acestui scenariu, că secvenţa de program a lui M_i lucrează cu un număr oarecare de variabile locale (de exemplu, 10), M_i ocupând, din acest punct de vedere, spaţiul maxim de memorie dintre toate ModX-urile aplicaţiei. Următorii paşi generali vor fi întâlniţi în operarea nucleului HARETICK, cu privire la execuţia celor două ModX-uri din Figura 9-7:

0) Aplicaţia a fost încărcată în sistem, structurile de date corespunzătoare au fost alocate şi iniţializate (PDAGT, PDT, zonele de cod şi "Data Out" pentru M_j şi M_i, *Heap*-ul pentru variabilele locale şi parametrii de intrare, zona de variabile globale, tabela de simboluri, etc.).

   Pentru zona de *Heap*, task-ul LOADER al nucleului va aloca, pentru toate ModX-urile aplicaţiei, o dimensiune de memorie de 13 locaţii: 10 cuvinte pentru

variabilele locale ale lui M_i şi 3 cuvinte pentru parametrii săi de intrare. Această decizie a LOADER-ului rezultă din ipoteza că M_i ocupă spaţiul maxim de memorie pentru variabilele locale şi parametrii de intrare, dintre toate ModX-urile aplicaţiei.

În cadrul contextului strict de execuţie, HSCD a planificat cele două ModX-uri, M_j şi M_i, pentru execuţie la momentele $t_{st,k}^{M_j}$, respectiv, $t_{st,m}^{M_i}$, unde k reprezintă perioada execuţiei lui M_j şi m perioada execuţiei lui M_i. Ca rezultat al planificării, Tabela de Dispatch va avea o configuraţie de tipul celei din Figura 9-8. Ca observaţie, momentul de start al ModX-ului M_i nu poate apărea mai devreme de terminarea execuţiei lui M_j, adică:

$$t_{st,m}^{M_i} \geq t_{st,k}^{M_j} + T_{ex}^{M_j} = t_{st,k}^{M_j} + \texttt{Mj.WCE}$$

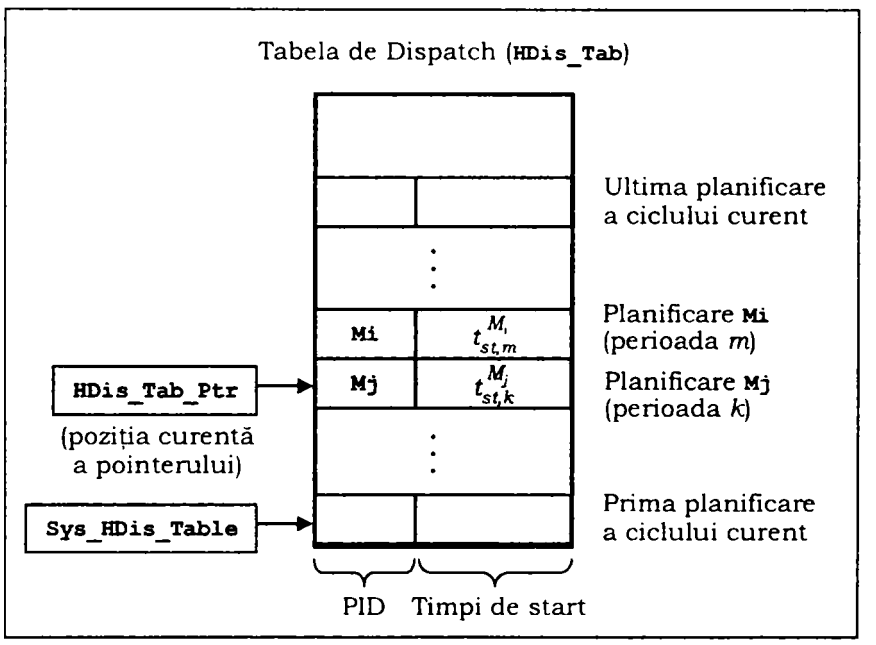

Figura 9-8. Configuraţia Tabelei de Dispatch pentru scenariul considerat

Presupunem că sistemul tocmai termină execuţia ModX-ului M_j. Prin urmare, pointerul Tabelei de Dispatch (`HDis_Tab_Ptr`) indică spre M_j (ModX-ul curent în execuţie).

1) Secvenţa de instrucţiuni a lui M_j se termină în mod obligatoriu cu apelarea componentei sufix a executivului TR al HARETICK, `HDIS_SUF` (vezi Subcapitolul 8.2.8 şi Capitolul 10). Instrucţiunea de apel la `HDIS_SUF` înlocuieşte astfel, instrucţiunile de tip `return` sau `RTS` ("ReTurn from Subroutine"), care încheie în mod uzual subrutinele de program (procedurile, funcţiile sau programele principale).

   Ca urmare, se activează HDIS, care actualizează pointerul Tabelei de Dispatch, `HDis_Tab_Ptr`, astfel încât acesta va indica spre M_i. De asemenea, `HDIS_SUF` verifică dacă mai este timp suficient, din momentul curent (al

terminării execuţiei efective a lui Mj) până la momentul $t_{st,m}^{M_i}$, pentru a reda controlul contextului lejer de execuţie. În caz contrar, `HDIS_SUF` iniţiază o buclă de aşteptare a întreruperii RTC pentru $t_{st,m}^{M_i}$.

2) La momentul $t_{st,m}^{M_i}$, RTC activează întreruperea de timer, având ca rezultat lansarea lui `HDIS_PRE` în execuţie. Acesta programează regiştrii de comparare ai timerelor RTC pentru momentul startării execuţiei ModX-ului ce urmează lui M_i ca planificare, în Tabela de Dispatch. În continuare `HDIS_PRE` verifică valoarea contorului de execuţii (`Mi.ECT`) al lui M_i, existând trei posibilităţi:

   - (2.1) `Mi.ECT = ECT_CONT`: ModX-ul M_i are specificat (momentan) un număr continuu de execuţii. Ca rezultat, HDIS nu va modifica acest parametru şi va lansa în execuţie pe M_i.
   - (2.2) `Mi.ECT` are o valoare finită. În acest caz, HDIS va decrementa cu o unitate contorul de execuţii şi apoi va lansa în execuţie pe M_i.
   - (2.3) `Mi.ECT = 0`: ModX-ul M_i este un *ModX fantomă* (vezi şi discuţia din secţiunea anterioară). Ca rezultat, HDIS lasă nemodificat contorul de execuţii *şi nu va lansa în execuţie* pe M_i, ci va apela pe `HDIS_SUF` (ca şi când s-ar fi terminat execuţia lui M_i).

3) *Odată activat ModX-ul M_i, înainte de a începe execuţia instrucţiunilor din zona de cod proprie, acesta va copia în locaţiile din zona* Heap *corespunzătoare parametrilor săi de intrare, valorile parametrilor de ieşire ai ModX-urilor faţă de care* M_i *prezintă dependenţe de date. Referirea parametrilor de ieşire se face pe baza tabelei de simboluri a HARETICK, care indică pentru fiecare parametru, adresa din zona "Data Out" corespunzătoare ModX-ului de care aparţine.*

   Astfel, în scenariul de faţă, M_i va copia în cele trei locaţii de memorie asignate de LOADER în Heap pentru parametrii săi de intrare (`Heap.param1`, `Heap.param2` şi `Heap.param3`), valorile `Mj.param1`, `Mj.param2`, respectiv, `Mi.param3`. Cele trei valori sunt regăsite de M_i, prin referirea la zona `Mj.DOut` pentru primii doi parametri şi, pentru cel de-al treilea, la zona `Mi.DOut`, utilizând adresele corespondente din tabela de simboluri a HARETICK.

Observaţie

Motivaţia acestui mecanism de copiere a parametrilor de intrare ai unui ModX înaintea execuţiei secvenţei sale propriu-zise de cod, derivă din necesitatea obţinerii unei predictibilităţi maxime de operare a sistemului HARETICK.

Cum numărul parametrilor de intrare/ieşire diferă de la un ModX la altul într-un mod transparent executivului TR al sistemului, a-l implementa pe acesta să efectueze operaţiile de copiere a valorilor momentane, sau utilizarea unei stive, reprezintă soluţii extrem de ineficiente şi surse de impredictibilitate majoră în cadrul sistemului.

Pe de altă parte, includerea acestor operaţii în codul ModX-urilor, cu calcularea corespunzătoare a duratei maxime de execuţie (`Mi.WCE`) astfel încât să cuprindă şi timpii necesari copierii parametrilor, conferă sistemului o mai mare flexibilitate, şi, mai ales, *nu îi afectează predictibilitatea.*

Operaţiile de copiere a valorilor parametrilor de intrare pentru un ModX se pot specifica, fie explicit – prin scrierea secvenţei respective de instrucţiuni chiar la începutul codului program al ModX-ului, fie implicit, de către compilator (de exemplu) – prin inserarea instrucţiunilor la începutul secvenţei de cod compilat al ModX-ului.

Din cele de mai sus se desprinde şi motivaţia structurilor de memorie care reprezintă zonele cu parametrii de ieşire ai fiecărui ModX, "Data Out", propuse în proiectul HARETICK. Mai mult, cu ajutorul acestor structuri se garantează păstrarea valorilor pe care un ModX doreşte să le comunice (altor ModX-uri, sau lui însuşi), de la execuţia curentă, până la următoarea execuţie (perioadă) a sa.

4) Execuţia lui M_i se continuă, după copierea parametrilor săi de intrare în Heap, cu secvenţa program proprie, stocată în zona de cod (`Mi.CODE`) de către LOADER, şi referită în cadrul înregistrării corespunzătoare din tabela descriptorilor de procese (PDT).

   La sfârşitul execuţiei, parametrul `param3`, alocat în zona "Data Out" a lui M_i (`Mi.param3`) va conţine noua valoare, rezultată în urma calculelor efectuate de M_i. Execuţia lui M_i se finalizează prin apelarea directă a lui `HDIS_SUF`.

9.6 Concluzii

Importanţa coordonatei temporale pentru sistemele TR ce implementează aplicaţii critice este subliniată în cadrul nucleului HARETICK prin utilizarea şi gestionarea a trei structuri distincte de reprezentare a timpului: `Sys_AbsTime` pentru timpul absolut sistem, ceasul timp-real (RTC) bazat pe contoare ale timerelor procesor, şi timpul de planificare, `HScd_TSched`. Cele trei structuri sunt sincronizate între ele cu ajutorul ModX-urilor sistem (HDIS şi HSCD), şi sunt în strictă dependenţă de tactul intern al procesorului ţintă.

Întreruperea de comparare RTC este, în stadiul actual de dezvoltare a sistemului, singura sursă admisă de întreruperi, fiind de asemenea predictibilă. Întreruperea RTC apelează în mod direct executivul HARETICK (mai precis, componenta prefix, HDIS_PRE), pentru lansarea în execuţie a ModX-ului planificat la momentul apariţiei întreruperii.

În cadrul nucleului HARETICK, ModX-urile sunt reprezentate cu ajutorul unor structuri şi mecanisme specifice:

- "PDAGT" (Program Directed Acyclic Graph Table), tabela ce conţine graful direcţionat, aciclic, de program al aplicaţiei;
- Zona de cod ("CODE") a ModX-ului;
- Zona parametrilor de ieşire ("Data Out");
- Zona variabilelor globale ale aplicaţiei;

- Zona variabilelor locale şi ai parametrilor de intrare ai ModX-ului în execuţie ("Heap");
- Tabela de simboluri;
- Tabela descriptorilor de procese ("PDT").

ModX-urile sunt executate în cadrul contextului strict al HARETICK, fiind planificate de către ModX-ul sistem HSCD. Activarea şi terminarea ModX-urilor este gestionată de către executivul TR al nucleului, HDIS.

La începutul execuţiei fiecărui ModX, are loc în mod obligatoriu operaţia de copiere a valorilor parametrilor de ieşire, din zonele "Data Out" ale ModX-urilor de care depinde cel curent, în locaţiile parametrilor de intrare, prevăzute în "Heap". Timpii de execuţie a instrucţiunilor implicate de această operaţie se vor adăuga la WCET-ul ModX-ului.

Comunicaţia inter-task-uri este realizată prin transmiterea parametrilor de intrare/ieşire, cu mecanismul descris anterior. Datorită structurii şi reprezentării ModX-urilor (care, practic, implementează operaţii atomice), şi a mecanismului de comunicaţie dintre ele, controlul şi sincronizarea acceselor la resurse concurente nu mai reprezintă o problemă. Un alt aspect interesant legat de acest subiect îl constituie facilitatea de transmitere a stării unui ModX, de la o execuţie a sa la alta, prin intermediul unui acelaşi parametru care este şi de intrare, şi de ieşire.

La sfârşitul execuţiei fiecărui ModX, se găseşte instrucţiunea de apel la HDIS, care, practic înlocuieşte în mod echivalent instrucţiunile utilizate uzual în acest caz (`return`, `exit`, `end`, `RTS`, etc.).

Din punctul de vedere al cerinţelor impuse de arhitectura nucleului HARETICK, calcularea timpului maxim de execuţie a ModX-urilor (WCET) implică următoarele elemente:

- durata de execuţie a celei mai lungi căi de program din codul propriu-zis al ModX-ului;
- durata secvenţei de instrucţiuni, de la începutul execuţiei ModX-ului, care efectuează operaţiile de copiere a parametrilor de ieşire;
- durata instrucţiunii de apel la `HDIS_SUF`, de la sfârşitul secvenţei de cod a ModX-ului.

10 Funcţionarea de principiu a sistemului

10.1 Contextele de execuţie sistem şi comutarea acestora

Nucleul HARETICK pune la dispoziţia aplicaţiilor TR o serie de structuri şi mecanisme, menite să maximizeze predictibilitatea operării şi să garanteze respectarea termenelor stricte impuse task-urilor în faza de proiectare/specificare:

- ModX-urile sunt modele ale task-urilor TR stricte, periodice, modulare;
- Planificarea ModX-urilor se efectuează într-o manieră complet predictibilă, aplicând algoritmi non-preemptivi, special concepuţi;
- Execuţia ModX-urilor are loc într-un context strict, conform planificării, fiind controlată de către executivul TR al nucleului;
- Operarea executivului TR (HDIS) se bazează pe întreruperea de ceas sistem (RTC) – singura întrerupere existentă în sistem.

Contextul de execuţie descris mai sus este denumit *contextul strict* al nucleului HARETICK (sau, *contextul HRT* – Hard Real-Time), şi reprezintă modalitatea de bază sub care nucleul tratează operarea task-urilor TR stricte ale unei aplicaţii.

Pe de altă parte, o serie de considerente au determinat introducerea şi implementarea unui al doilea context de execuţie în cadrul sistemului HARETICK:

- Majoritatea aplicaţiilor TR, indiferent de tipul sau complexitatea acestora, conţin, pe lângă task-urile TR stricte, şi task-uri cu specificaţii lejere de comportare în timp, sau chiar task-uri pentru care coordonata temporală nu e deloc importantă.
- Planificarea şi execuţia în context strict, non-preemptiv a ModX-urilor este foarte puţin flexibilă şi implică o eficienţă scăzută de operare a sistemului.
- Planificarea şi execuţia ModX-urilor se bazează pe timpii maximi de execuţie ai acestora, care se determină într-o manieră pesimistă (calea de execuţie cu durata cea mai mare din secvenţa de instrucţiuni a ModX-ului).
- Existenţa ModX-urilor fantomă (vezi capitolul anterior).

Pe baza acestor considerente, a fost introdus un context de execuţie complementar cu cel strict, anume *contextul lejer* al nucleului HARETICK (sau, *contextul SRT* – Soft Real-Time). În cadrul contextului lejer, sunt planificate şi executate task-urile sistem şi ale aplicaţiei, care nu impun cerinţe stricte de comportare temporală, şi pentru care garantarea respectării termenelor nu reprezintă o necesitate.

Task-urile lejere, notate cu L_i (spre deosebire de ModX-uri, care sunt notate cu M_i), se execută în regim preemptiv, în intervalele de timp rămase libere din cadrul contextului strict. Planificarea task-urilor lejere este efectuată de către un task sistem,

SSCD (Soft real-time Scheduler – la rândul lui, un task lejer), pe baza unor algoritmi ce utilizează priorităţi şi mecanisme de tip *time-sharing*.

Observaţie

Contextul strict de execuţie poate fi privit ca o "întrerupere prelungită" a celui lejer, conform unei planificări riguroase, stricte, analizată apriori din punctul de vedere al fezabilităţii (analiză "offline").

Un alt mod de a vedea problema este acela că, deasupra contextului lejer (reprezentând modul de operare tradiţional, preemptiv, al sistemelor de calcul) se poate implementa, ca o facilitate de execuţie cu garantarea respectării termenelor stricte şi cu predictibilitate maximă, contextul strict, a cărui întrerupere (RTC) este cea mai prioritară în sistem. În plus, pe durata execuţiei în contextul strict, celelalte întreruperi sunt suspendate.

Componenta sistem care asigură comutarea operării între cele două contexte de execuţie, este executivul TR al HARETICK, HDIS. Principiile operaţiilor de comutare între contextul strict şi cel lejer, aplicate de HDIS, sunt:

- În momentul apariţiei unei întreruperi RTC, executivul activează contextul strict de execuţie, lansând ModX-ul planificat la acel moment;
- După terminarea execuţiei unui ModX (sau în cazul unui ModX fantomă), HDIS verifică intervalul de timp rămas disponibil până când este planificat următorul ModX, şi în caz afirmativ, restaurează contextul de execuţie al ultimului task lejer întrerupt de sistem.

O exemplificare detaliată a modului de operare al HARETICK, cu cele două contexte de execuţie, este prezentată în Figura 10-1. Utilizând aceeaşi axă a timpului, în figură sunt reprezentate graficele de operare ale contextului strict (HRT) şi lejer (SRT), din punctul de vedere al planificării (partea superioară) şi al execuţiei efective în sistem (partea inferioară).

Notaţiile utilizate în Figura 10-1 pentru componentele HARETICK sunt după cum urmează:

- "P-HSCD" – execuţia în regim de pre-planificare a planificatorului HSCD;
- "PD" – componenta prefix a executivului TR, HDIS_PRE;
- "SD" – componenta sufix a executivului TR, HDIS_SUF;
- "SS" – planificatorul task-urilor contextului lejer de execuţie, SSCD;
- "Li" – task-ul lejer *i* al sistemului/aplicaţiei;
- "Mi" – ModX-ul *i* al sistemului/aplicaţiei;
- "t_{Sk}" – momentul *k* al planificării (scheduling time), echivalent cu momentul de start sau de terminare a planificării unui ModX.

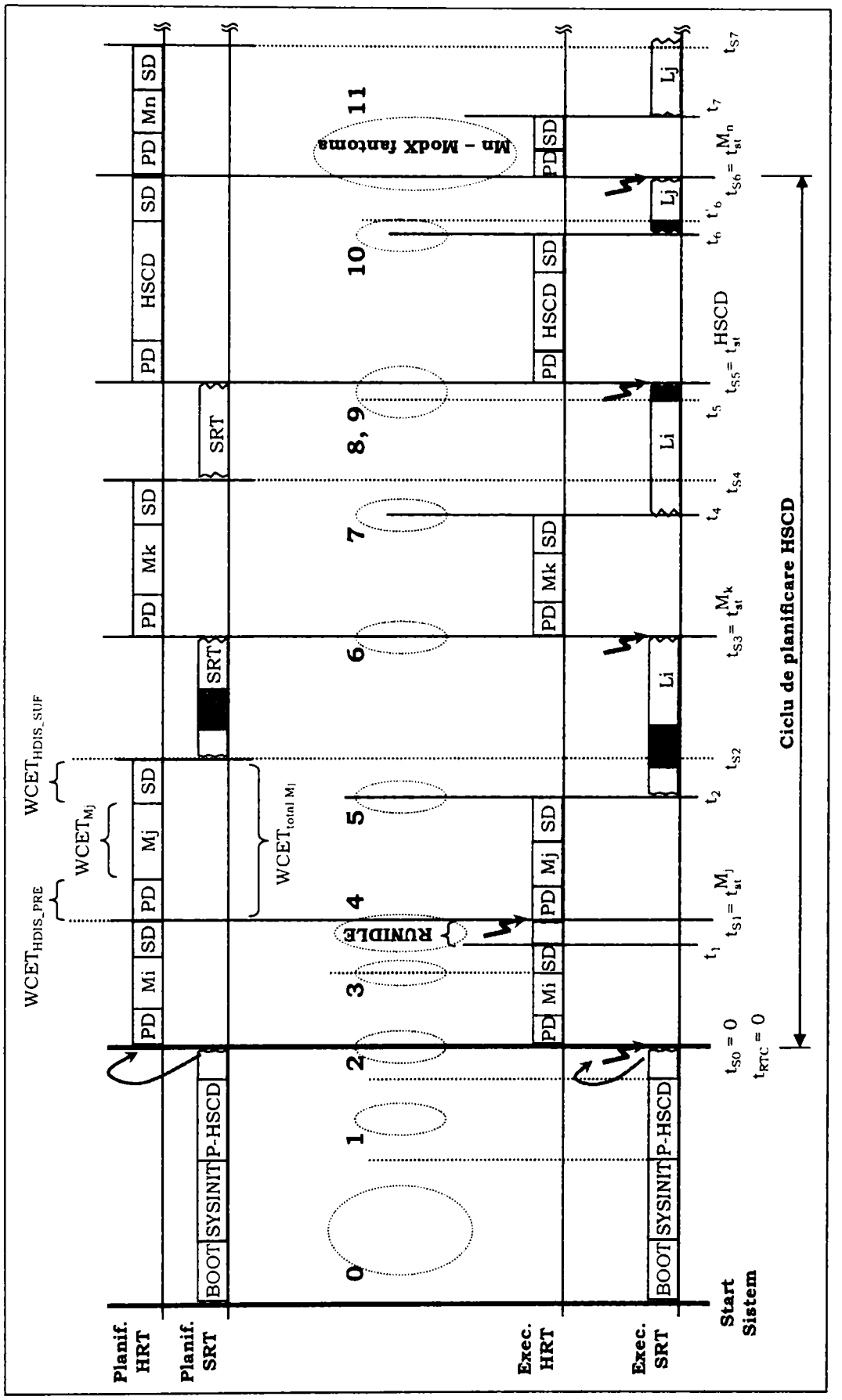

Figura 10-1. Planificarea şi execuţia task-urilor în HARETICK

Pe parcursul operării nucleului HARETICK, se remarcă următoarele evenimente (situaţii) speciale, marcate în ordinea corespunzătoare pe diagrama din Figura 10-1:

0) Iniţial, după startarea sistemului, rulează secvenţa BOOT care, după încărcarea celorlalte componente de bază ale HARETICK în memorie, activează task-ul lejer SYSINIT.

SYSINIT efectuează iniţializările necesare, după care lansează planificatorul contextului strict, HSCD, în regim de *pre-planificare*.

1) Execuţia planificatorului în regim de pre-planificare (P-HSCD) este similară din punct de vedere funcţional cu o execuţie normală a HSCD, singura diferenţă fiind că în acest caz, contextul strict al HARETICK nu este încă activat. Aşadar, P-HSCD se execută în context lejer (SRT).

Odată lansat în execuţie, HSCD realizează planificarea non-preemptivă a setului de ModX-uri din sistem, pentru un ciclu. Ciclul de planificare este definit ca seria ordonată în timp a planificărilor de forma

{<*indentificator ModX*>, <*moment start execuţie*>}

şi care este încadrată de două execuţii consecutive ale HSCD.

Pentru cazul considerat în exemplul nostru, după execuţia P-HSCD, Tabela de Dispatch va conţine patru planificări, ce definesc practic primul ciclu de planificare: trei execuţii de ModX-uri şi execuţia HSCD care încheie ciclul (şi, totodată, startează ciclul următor). Astfel, la momentul iniţial al contextului HRT, t_0, este planificată execuţia ModX-ului M_i, la terminarea căruia (momentul t_{S_1}), va fi lansat în execuţie ModX-ul M_j, a cărui execuţie va dura până la momentul t_{S_2}. Mai departe, la momentul t_{S_3}, va fi executat ModX-ul M_k, care se termină la t_{S_4}. Execuţia lui HSCD la t_{S_5} termină ciclul curent de planificare şi va iniţia următorul ciclu (aici – al doilea).

Figura 10-2 ilustrează modul în care P-HSCD modifică conţinutul Tabelei de Dispatch, de la starea iniţială (listă vidă, setată de SYSINIT), la starea corespunzătoare terminării algoritmului de planificare pentru ciclul curent.

După terminarea execuţiei P-HSCD, este apelat din nou task-ul SYSINIT, care activează contextul HRT al HARETICK, prin setarea ceasului timp-real (RTC) al sistemului pe 0 ($t_0 = 0$) şi a registrelor de comparare ale timerelor RTC, de asemenea pe 0. Apoi este activată întreruperea RTC.

Observaţie

Algoritmii de planificare non-preemptivă consideraţi pentru implementarea lui HSCD operează pe baza timpului maxim de execuţie al ModX-urilor (WCET). Aşa cum se vede şi în Figura 10-1, durata maximă de execuţie a oricărui ModX include şi WCET al executivului TR al sistemului, HDIS (cu cele două componente ale sale – prefix şi sufix).

Aceasta este o altă cerinţă importantă impusă de implementarea nucleului HARETICK asupra calculului timpilor WCET ai ModX-urilor.

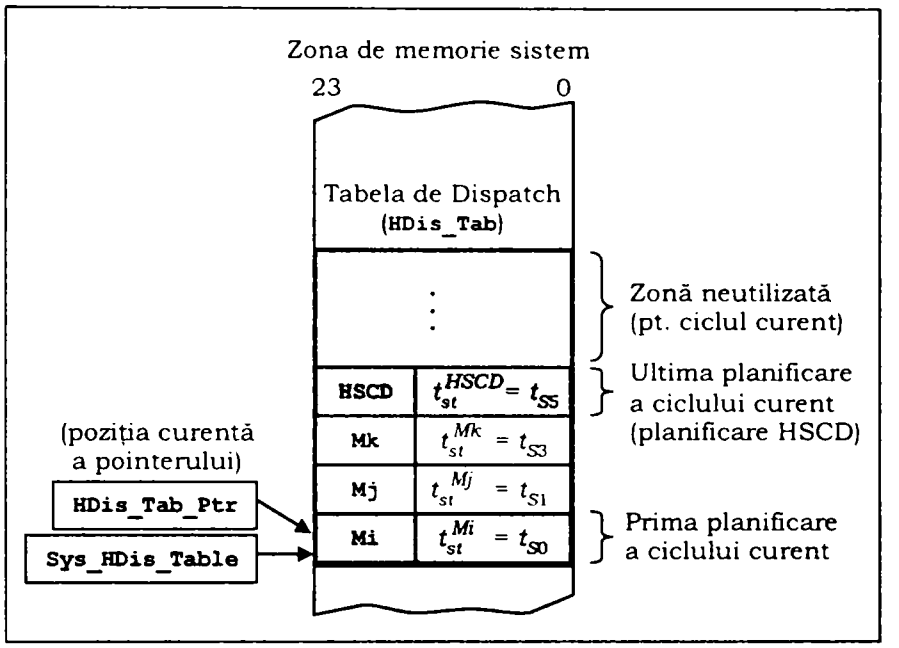

Figura 10-2. Conţinutul Tabelei de Dispatch după execuţia lui P-HSCD

2) În momentul activării întreruperilor RTC, task-ul SYSINIT este întrerupt şi se lansează (se comută) contextul strict de execuţie al HARETICK, prin apelarea componentei prefix a executivului, HDIS_PRE.

HDIS_PRE încarcă din Tabela de Dispatch (de la poziţia curentă a pointerului) informaţia legată de execuţia ModX-ului ce urmează a fi lansat în execuţie la momentul curent. În scenariul discutat, este vorba despre M_i, planificat la momentul $t_{S_0} = t_{st}^{M_i} = 0$.

Înainte de lansarea în execuţie a lui M_i, HDIS_PRE citeşte momentul de start al următoarei planificări (înregistrarea următoare din Tabela de Dispatch), în cazul nostru, $t_{S_1} = t_{st}^{M_j}$, şi programează regiştrii de comparare ai timerelor RTC pentru generarea unei întreruperi la acel moment.

3) La terminarea execuţiei ModX-ului curent (M_i, care aici, are loc la instanţa t_1, anterioară momentului considerat la planificare), se va apela componenta sufix a executivului TR al HARETICK, HDIS_SUF.

HDIS_SUF avansează pointerul în Tabela de Dispatch (în acest punct, pointerul va indica spre ModX-ul M_j), şi determină intervalul de timp rămas disponibil (din momentul curent) până la momentul planificării lui M_j.

4) Situaţia reprezentată de evenimentul (4) în Figura 10-1, este singurul caz în care executivul HDIS însuşi este întrerupt de către RTC. Este vorba de starea specială a lui HDIS_SUF, denumită "RunIdle", şi care reprezintă practic o buclă de aşteptare a întreruperii, implementată de HDIS_SUF.

În faza de calculare a timpului disponibil (din momentul curent) până la momentul planificării lui M_j, HDIS_SUF se bazează pe o evaluare ce ţine cont de o serie de parametri de eficienţă a operării sistemului. Se apreciază astfel,

dacă timpii consumaţi de restaurarea şi salvarea contextului (a stării procesor) pentru task-ul lejer întrerupt, sunt suficient de mici comparativ cu timpul rămas pentru continuarea execuţiei acestuia, pentru ca operaţia de comutare a contextelor de execuţie să fie eficientă.

La momenul t_1, această evaluare determină decizia intrării lui HDIS_SUF în starea "RunIdle", adică de a nu comuta contextul strict de execuţie. Se va aştepta astfel, apariţia întreruperii RTC, care declanşează execuţia următorului ModX (M_j).

5) La momentul t_2, se termină execuţia lui HDIS_SUF corespunzătoare ModX-ului M_j. Intervalul de timp de la t_2, la $t_{S_3} = t_{st}^{M_k}$ (momentul planificării lui M_k), este suficient de mare pentru a permite comutarea contextului strict în contextul lejer de execuţie.

Principala operaţie realizată la activarea contextului SRT de către HDIS este restaurarea contextului de execuţie al ultimului task lejer întrerupt în sistem. În cazul de faţă, este vorba de task-ul SYSINIT, care îşi va continua execuţia cu iniţializarea celorlalte task-uri lejere din sistem, după care va lansa planificatorul contextului SRT de operare al HARETICK. SSCD va lansa, la rândul lui, în execuţie task-ul lejer L_i.

6) Contextul lejer de execuţie (task-ul L_i) este întrerupt de către RTC, la momentul planificării ModX-ului M_k, $t_{S_3} = t_{st}^{M_k}$. Rutina de tratare a întreruperii apelează executivul TR (HDIS_PRE), care salvează prima dată contextul de execuţie al task-ului lejer L_i.

7) Componenta sufix a lui HDIS este apelată la terminarea ModX-ului M_j, când se va efectua comutarea contextelor. HDIS_SUF restaurează contextul de execuţie a task-ului lejer întrerupt ultima dată (L_i), iar acesta îşi va continua execuţia.

8, 9) Întreruperea RTC pentru lansarea în execuţie a următorului ModX (aici, HSCD) apare în timpul execuţiei planificatorului task-urilor lejere, SSCD. HDIS va proceda la salvarea contextului de execuţie al SSCD, la fel ca şi cu celelalte task-uri lejere din sistem.

10) Restaurarea contextului de execuţie al lui SSCD are loc la momentul t_6, după care acesta îşi continuă operarea, lansând în execuţie un alt task lejer, L_j.

11) Cazul în care executivul HDIS constată că ModX-ul care urmează a fi lansat în execuţie (conform planificării) este un ModX fantomă.

Înainte de lansarea în execuţie a unui ModX, HDIS_PRE verifică valoarea contorului de execuţie al ModX-ului (`Mn.ECT`), din tabela descriptorilor de procese (PDT). Dacă `Mn.ECT` = 0, HDIS_PRE lasă contorul nemodificat, şi în locul lansării în execuţie a lui M_n, va apela componenta sufix, HDIS_SUF.

10.2 Stările ModX-urilor

Operarea ModX-urilor în cadrul sistemului HARETICK poate fi descrisă printr-un set de cinci stări:

(1) Starea NOP ("No Operation").

Starea NOP descrie ModX-urile care nu sunt încărcate în cadrul nucleului HARETICK la un moment dat, ci se află în stadiu de dezvoltare şi analiză în cadrul mediului integrat INVERTA, pe un calculator gazdă. Sistemul HARETICK nu are practic cunoştinţă despre aceste ModX-uri, ele neinfluenţând în nici un fel operarea curentă a nucleului de pe platforma ţintă.

Ca observaţie, remarcăm faptul că ModX-urile aparţinând nucleului HARETICK (de exemplu, HSCD, HDIS, componentele stricte ale lui DATALINK, etc.) nu se vor găsi niciodată în starea NOP, ele fiind încărcate pe platforma ţintă odată cu restul nucleului HARETICK în timpul operaţiei de BOOT. Doar ModX-urile aplicaţiilor ce vor fi încărcate în sistem la un moment dat, pot să se găsească în starea NOP, înaintea încărcării acestora.

(2) Starea RDY ("Ready").

Odată ce o aplicaţie oarecare a fost încărcată pe platforma ţintă pe care rulează nucleul HARETICK, ModX-urile acesteia vor intra în starea RDY ("gata de planificare").

Un ModX în starea RDY este complet identificat şi reprezentat în cadrul nucleului, având alocate majoritatea resurselor necesare execuţiei:

- identificare în cadrul aplicaţiei de care aparţine, prin tabela PDAGT;
- caracterizare completă prin parametrii săi din tabela PDT;
- alocarea zonei proprii de cod în memoria program a sistemului;
- alocarea zonei parametrilor de ieşire ("Data Out") în memoria de date;
- alocarea zonei cu variabilele globale ale aplicaţiei şi identificarea celor utilizate de ModX;
- identificarea variabilelor şi parametrilor utilizaţi, prin intermediul tabelei de simboluri a aplicaţiei.

(3) Starea "SCD" ("Scheduled").

ModX-urile din starea SCD aparţin planificării din cadrul ciclului curent, ca rezultat al aplicării algoritmilor de planificare strictă, non-preemptivă, implementaţi de HSCD, asupra setului de ModX-uri sistem şi ale aplicaţiei.

ModX-urile din starea SCD se regăsesc în cadrul Tabelei de Dispatch, rezultând că la un moment dat (într-un ciclu de planificare), există un număr bine determinat de ModX-uri în starea SCD.

(4) Starea "RUN" ("În execuţie").

În starea RUN se poate găsi, la un moment dat, un singur ModX. Acestuia i se alocă restul resurselor sistem necesare execuţiei efective pe platforma ţintă:

- zona "Heap" a sistemului, alocată exclusiv pentru parametrii de intrare şi variabilele locale ale ModX-ului, pe toată durata execuţiei sale, şi numai atunci;
- procesorul sistemului, care execută în regim non-preemptiv secvenţa de instrucţiuni a ModX-ului.

(5) Starea "GST" ("Ghost").

Starea GST este starea în care se află la un moment dat un ModX fantomă. ModX-ul fantomă este caracterizat prin faptul că, deşi a fost planificat de către HSCD, în momentul lansării în execuţie are contorul de execuţii nul.

ModX-urile fantomă nu sunt lansate în execuţie efectivă de către nucleu (adică de către executivul TR al nucleului, HDIS), deci nu li se alocă resursele suplimentare necesare execuţiei, ca în cazul stării RUN.

Trecerea unui ModX oarecare dintr-o stare în alta are loc în urma îndeplinirii unor condiţii specifice, şi are loc prin intermediul unui set distinct de operaţii. Figura 10-3 ilustrează cele cinci stări posibile, împreună cu configuraţia resurselor sistemului pentru fiecare stare în parte. Figura prezintă de asemenea, şi modurile posibile de trecere a ModX-urilor dintr-o stare în alta, precum şi task-urile sistem implicate în aceste operaţii.

Încărcarea unei aplicaţii oarecare în sistem este efectuată de către task-ul LOADER, prin intermediul componentei sistem DATALINK. LOADER-ul este responsabil de alocarea şi completarea structurilor necesare reprezentării aplicaţiei şi a ModX-urilor acesteia în cadrul nucleului HARETICK: tabela PDAGT, tabela PDT, tabela de simboluri, zonele de cod şi "Data Out" pentru fiecare ModX, zona variabilelor globale, etc.

Setul de operaţii ce corespunde trecerii ModX-urilor din starea NOP în RDY este denumit "load".

Descărcarea (eliminarea) unei aplicaţii din sistemul HARETICK este realizată tot de task-ul LOADER, eliberându-se resursele sistem utilizate de nucleu la reprezentarea şi gestionarea aplicaţiei. Setul de operaţii corespunzător trecerii unui ModX din starea RDY în starea NOP este denumit "unload".

Planificatorul contextului strict de execuţie, HSCD, aplică algoritmul de planificare non-preemptivă special conceput pentru sistemul HARETICK, asupra setului de ModX-uri aflate în starea RDY. În cadrul unui ciclu de planificare, va fi selectat un număr bine definit de ModX-uri, cu care se construieşte (completează) Tabela de Dispatch. Ca urmare, starea ModX-urilor planificate se schimbă, din RDY în SCD.

Ca observaţie, menţionăm că un anumit ModX poate apărea în cadrul mai multor înregistrări din Tabela de Dispatch, fiind astfel planificat pentru execuţie de mai multe ori în ciclul curent de planificare. De exemplu, un ModX cu perioadă mai mică are o frecvenţă mai mare de apariţii în cadrul Tabelei de Dispatch, comparativ cu un ModX cu perioadă mai mare.

Setul de operaţii prin care un ModX trece din starea RDY în SCD este denumit "schedule". Cum se observă şi din Figura 10-3, arhitectura nucleului HARETICK nu permite tranziţia inversă a ModX-urilor, anume din SCD, direct în RDY. Cu alte

cuvinte, odată ce un ModX a fost planificat pentru ciclul curent de planificare, el va fi preluat în continuare de executivul TR al nucleului, HDIS, în vederea lansării în execuţie la momentul specificat în planificare.

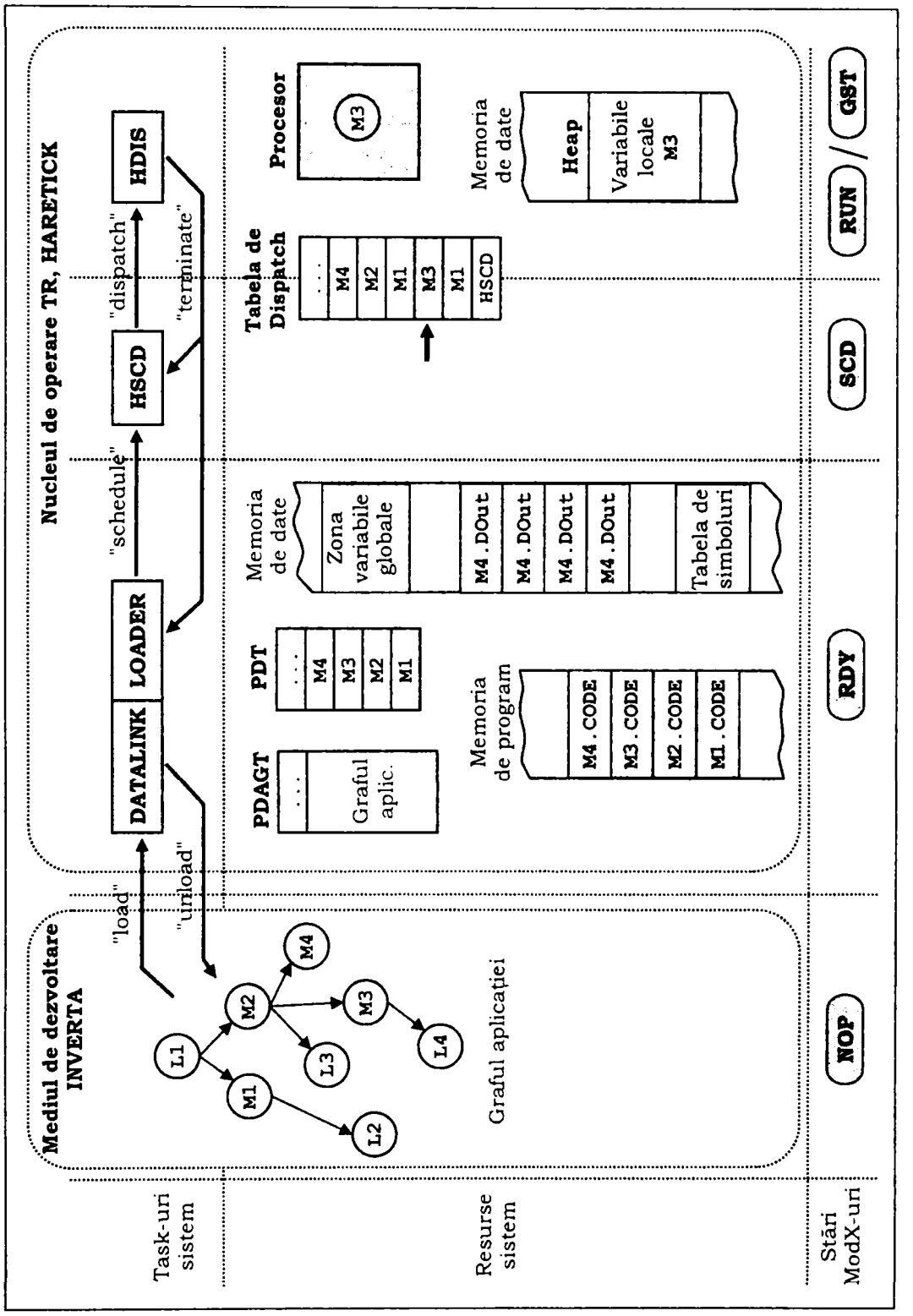

Figura 10-3. Stările ModX-urilor şi operarea nucelului HARETICK

Tabelul 10-1. Sinteză a stărilor ModX-urilor şi a tranziţiilor între stări

Tranziţie stări	Denumire	Semnificaţie
NOP → RDY	"load"	Încărcarea unei aplicaţii dezvoltate şi analizate în mediul integrat INVERTA de pe un calculator gazdă, pe platforma ţintă ce rulează sub nucleul TR de operare HARETICK.
RDY → NOP	"unload"	Descărcarea (eliminarea) unei aplicaţii din sistemul HARETICK şi eliberarea resurselor sistem alocate pentru reprezentarea şi gestionarea ei.
RDY → SCD	"schedule"	Includerea unui subset de ModX-uri ale sistemului (aplicaţiei) în ciclul curent de planificare, în vederea execuţiei pe sistemul HARETICK.
SCD → {RUN, GST}	"dispatch"	Preluarea de către executivul HDIS a unui ModX inclus în ciclul curent de planificare, în vederea execuţiei. În cazul în care contorul său de execuţii este nenul, ModX-ul va fi executat efectiv (starea RUN), altfel, se va considera doar că a fost executat (starea GST).
{RUN, GST} → {RDY, SCD}	"terminate"	Terminarea execuţiei (efective – RUN, sau fictive – GST) a unui ModX, urmată de revenirea acestuia în starea "gata de planificare" (RDY) sau "planificat şi gata de execuţie" (SCD), în cazul în care ciclul curent de planificare prevede mai multe execuţii ale ModX-ului.

ModX-urile planificate de către HSCD sunt preluate pe rând de către executivul HDIS, prin baleierea Tabelei de Dispatch, în vederea lansării în execuţie pe baza mecanismului de întreruperi RTC. La un moment dat, ceasul timp-real al sistemului va genera o întrerupere, semnalând instanţa de timp pentru care a fost planificată execuţia ModX-ului curent referit în Tabela de Dispatch. Rutina de tratare a întreruperilor apelează executivul HDIS, care verifică valoarea contorului de execuţii a ModX-ului curent. În cazul în care contorul de execuţii este nul, ModX-ul devine un ModX fantomă, trecând astfel în starea GST. HDIS nu îl va lansa în execuţie efectivă, ci va trece la următorul ModX din tabelă.

Prin urmare, ModX-urile din starea GST sunt considerate de sistem ca "fiind executate", fără a fi lansate efectiv în execuţie (de aici şi denumirea de "ModX fantomă").

În cazul în care contorul de execuţii al ModX-ului curent ce urmează a fi executat este nenul, executivul HDIS îi alocă restul resurselor sistem necesare execuţiei acestuia (zona "Heap" pentru parametrii de intrare şi variabilele locale, şi procesorul sistemului). ModX-ul trece astfel în starea RUN, ocupând exclusiv procesorul sistemului pentru execuţia în regim non-preemptiv a secvenţei proprii de instrucţiuni.

Setul de operaţii implicate în tranziţia ModX-urilor din starea SCD în RUN sau GST, este denumit "dispatch".

După execuţia (efectivă sau "fantomă") a unui ModX aflat în starea RUN, respectiv GST, acesta poate reveni în starea RDY sau SCD. În primul caz, ModX-ul care tocmai a fost executat şi-a terminat şi planificările din ciclul curent de planificare, adică nu mai există o altă înregistrare corespunzătoare aceluiaşi ModX în Tabela de Dispatch. Cea de-a doua variantă, presupune contrariul celor de mai sus, adică ModX-ul respectiv se mai regăseşte în cadrul tabelei, urmând a mai fi executat în acelaşi ciclu de planificare. El rămâne astfel, în starea de ModX planificat (SCD).

Setul operaţiilor implicate de terminarea execuţiei ModX-urilor şi trecerea lor din starea {RUN, GST} în starea {SCD, RDY}, este denumit "terminate".

Tabelul 10-1 oferă o sinteză a celor discutate până acum în legătură cu stările ModX-urilor sistemului HARETICK şi tranziţiile posibile între stări.

10.3 Executivul TR al HARETICK: HDIS

10.3.1 Caracteristici generale

Executivul TR al HARETICK, HDIS (Hard real-time Dispatcher), este o componentă esenţială a nucleului şi are principala sarcină de a lansa în execuţie ModX-urile sistem şi ale aplicaţiei aflate în starea SCD ("planificat"), şi de a termina execuţia acestora, în contextul strict de execuţie al nucleului HARETICK. HDIS implementează astfel mecanismele necesare comutării celor două tipuri de contexte de execuţie existente în sistem: contextul strict (contextul HRT – Hard Real-Time) în cadrul căruia se execută în regim non-preemptiv ModX-urile sistemului, şi contextul lejer (contextul SRT – Soft Real-Time) ce corespunde execuţiei în regim preemptiv a task-urilor lejere.

Operarea HDIS se face într-o manieră complet predictibilă, bazată pe următoarele considerente:

- HDIS este apelat de către rutina de tratare a întreruperii de ceas timp-real (RTC), la momente de timp bine stabilite, conform planificării execuţiei task-urilor TR stricte în sistem;

- Durata maximă de execuţie a HDIS ($WCET_{HDIS}$) este calculată cu atenţie, luând în considerare toate situaţiile posibile de operare ale nucleului. Pe orice ramură de program s-ar afla la un moment dat execuţia codului HDIS, timpii maximi de execuţie pot fi determinaţi cu precizie, fără a depinde de structuri sau mecanisme dinamice (cum ar fi de exemplu, numărul curent de ModX-uri în sistem, numărul parametrilor de intrare sau de ieşire al acestora, etc.);
- Întreaga arhitectură a HARETICK şi a mecanismelor de reprezentare şi gestionare a ModX-urilor în cadrul nucleului, este concepută şi proiectată în aşa fel încât să faciliteze operarea predictibilă a executivului şi a întregului sistem. Sunt utilizate doar reprezentări statice şi operaţii precis limitate în timp şi spaţiu. În orice moment, sunt cunoscute cu precizie valorile maxime ale dimensiunilor structurilor folosite şi duratele maxime de execuţie ale operaţiilor.

Executivul HDIS are două componente principale: un prefix (HDIS_PRE) şi un sufix (HDIS_SUF), cu roluri bine determinate. Astfel, HDIS_PRE iniţiază execuţia ModX-ului planificat a fi lansat la un moment dat, şi este apelat de întreruperea RTC ce apare la acel moment în sistem. Odată activat, HDIS_PRE efectuează următoarele acţiuni:

- verifică dacă a fost întrerupt contextul lejer. În caz afirmativ, se efectuează salvarea contextului de execuţie lejeră a task-ului întrerupt într-o zonă dedicată;
- identifică ModX-ul ce urmează a fi lansat în execuţie la momentul curent, din Tabela de Dispatch;
- citeşte următoarea înregistrare din Tabela de Dispatch, corespunzând ModX-ului ce urmează ca planificare celui curent, şi extrage momentul de start a execuţiei acestuia;
- programează timerele ceasului de timp-real (RTC) pentru a genera întrerupere la momentul de start a ModX-ului următor;
- decrementează contorul de execuţii a ModX-ului curent (dacă acesta are valoare finită);
- trece ModX-ul curent din starea "planificat" ("SCD"), fie în starea "de execuţie" ("RUN") – dacă valoarea contorului de execuţie era nenulă înaintea decrementării, fie în starea " ModX fantomă" ("GST");
- dacă ModX-ul curent este în starea "RUN", îi cedează procesorul şi îl apelează pentru execuţie;
- dacă ModX-ul curent este în starea "GST", apelează direct componenta sufix a executivului, HDIS_SUF.

Pe de altă parte, componenta sufix a lui HDIS este apelată la terminarea execuţiei fiecărui ModX, chiar de către acesta. Odată activat, HDIS_SUF efectuează următoarele acţiuni:

- trece ModX-ul curent din starea "de execuţie" (RUN) sau "ModX fantomă" (GST), înapoi în starea "planificat şi gata de execuţie" (SCD) sau în "gata de planificare" (RDY);
- avansează referinţa (pointerul) de citire al Tabelei de Dispatch către ModX-ul următor;

- verifică dacă există suficient timp pentru a comuta contextul strict cu cel lejer, permiţând execuţia task-urilor SRT ale sistemului şi aplicaţiei. Dacă nu este timp suficient, HDIS va iniţia o buclă infinită ("RunIdle") în care va aştepta apariţia întreruperii RTC ce marchează evenimentul lansării în execuţie a ModX-ului următor. Dacă se decide comutarea controlului către contextul lejer, are loc restaurarea stării de execuţie (a contextului) task-ului lejer care a fost întrerupt.

Pe lângă cele două componente principale, executivul HDIS mai utilizează o subrutină denumită "TiLT" ("Time Log Tool"), pentru generarea unor scurte rapoarte legate de evenimentele lansării în execuţie şi terminării ModX-urilor. Evenimentele sunt înscrise într-o tabelă dedicată, împreună cu valorile instanţelor de timp corespunzătoare.

Structura de date de bază a executivului HDIS este Tabela de Dispatch (vezi şi Capitolul 9). Tabela de Dispatch (HDis_Tab) este o structură sistem cu un număr fix, bine stabilit de înregistrări (λ + 1), care specifică numărul maxim de planificări posibile pentru orice ciclu de planificare: λ planificări de ModX-uri normale (sistem sau ale aplicaţiei) şi o planificare de HSCD, care încheie ciclul curent şi iniţiază ciclul următor de planificare. HDIS accesează Tabela de Dispatch doar în citire, completarea înregistrărilor tabelei fiind efectuată de către planificatorul contextului strict de execuţie al HARETICK, HSCD.

Fiecare înregistrare a Tabelei de Dispatch conţine două câmpuri: (i) identificatorul ModX-ului, PID, şi (ii) momentul de start al execuţiei ModX-ului, conform planificării realizate de HSCD în cadrul ciclului curent de planificare. Tabela este definită prin adresa de bază (`Sys_HDis_Table`) şi de dimensiunea sa totală ($(\lambda + 1) \cdot$ *<dimensiunea unei înregistrări>*). Înregistrarea curentă în cadrul tabelei este referită printr-un pointer, `HDis_Tab_Ptr`.

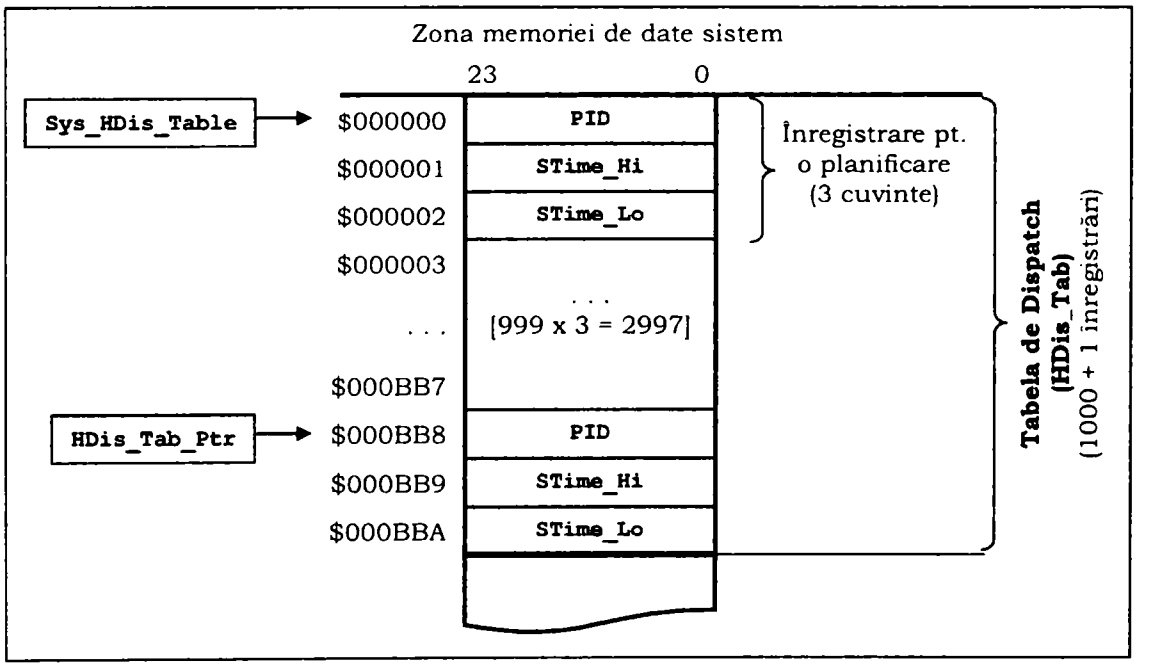

Figura 10-4. Implementarea curentă a Tabelei de Dispatch

Figura 10-4 ilustrează modul curent de implementare a Tabelei de Dispatch, pe arhitectura Motorola DSP56307. Fiecare înregistrare conţine un cuvânt procesor pentru identificarea ModX-ului (PID) şi două cuvinte (2·24 biţi) pentru exprimarea timpului de start al execuţiei ModX-ului (`STime_Hi:STime_Lo`). Structura de timp a înregistrărilor tabelei este compatibilă cu arhitectura ceasului timp-real (RTC) care e compusă din două timere cascadate (sau un timer şi o variabilă sistem). În acest fel se uşurează operaţiile de programare a regiştrilor de comparare ai RTC de către HDIS.

Dimensiunea totală a tabelei este de 1000 (înregistrări normale) + 1 (înregistrare specială), rezultând un număr de 3003 cuvinte de memorie. Prin urmare, un ciclu de planificare va cuprinde maxim 1001 execuţii, din care una corespunde planificatorului HSCD.

10.3.2 Elemente de implementare şi analiză a codului executivului

În continuare vom detalia pe rând funcţionarea celor trei componente ale executivului HDIS: HDIS_PRE, HDIS_SUF şi subrutina TiLT.

Separarea executivului în două componente, prefix şi sufix, a fost realizată ţinând cont de cele două ipostaze de apelare ale lui HDIS şi de modul în care acesta încadrează execuţia propriu-zisă a unui ModX. Componenta prefix este apelată de către întreruperea RTC şi iniţiază execuţia ModX-urilor, pe când componenta sufix este apelată de către ModX-uri, la terminarea execuţiei acestora.

Ramurile de procesare ale executivului HDIS mai trebuie să trateze o situaţie importantă: execuţia planificatorului contextului strict, HSCD. În cazurile în care ModX-ul curent ce urmează a fi lansat în execuţie este identificat în Tabela de Dispatch ca fiind HSCD, executivul nu poate citi din tabelă timpul de start pentru ModX-ul următor, în vederea programării RTC. Acest lucru se datorează faptului că înregistrările din tabelă (ce identifică planificări ale ModX-uri normale) urmează a fi completate chiar de către planificator, în timpul execuţiei sale.

Soluţia constă în faptul că, atunci când e vorba de lansarea în execuţie a HSCD, programarea timerelor RTC nu se mai efectuează de către executivul prefix, ci de către componenta sufix, după terminarea execuţiei planificatorului ce are ca rezultat completarea tabelei cu planificările noului ciclu.

Figura 10-5, continuată cu Figura 10-6, prezintă organigrama de operare a componentei prefix a executivului. HDIS_PRE este lansată în execuţie de către întreruperea RTC, anume de întreruperea de `Timer0`, în implementarea curentă a nucleului HARETICK pe arhitectura Motorola DSP56307. Fiind vorba de două timere cascadate ca bază de timp în sistem, `Timer0` şi `Timer1`, a apărut problema rezolvării unei întreruperi suplimentare – cea generată de `Timer1`. Acesta va genera întreruperi doar dacă între două planificări succesive de ModX-uri există un interval de timp ce depăşeşte capacitatea curentă de numărare rămasă disponibilă în `Timer0` (registrul de numărare al `Timer0` se va "da peste cap").

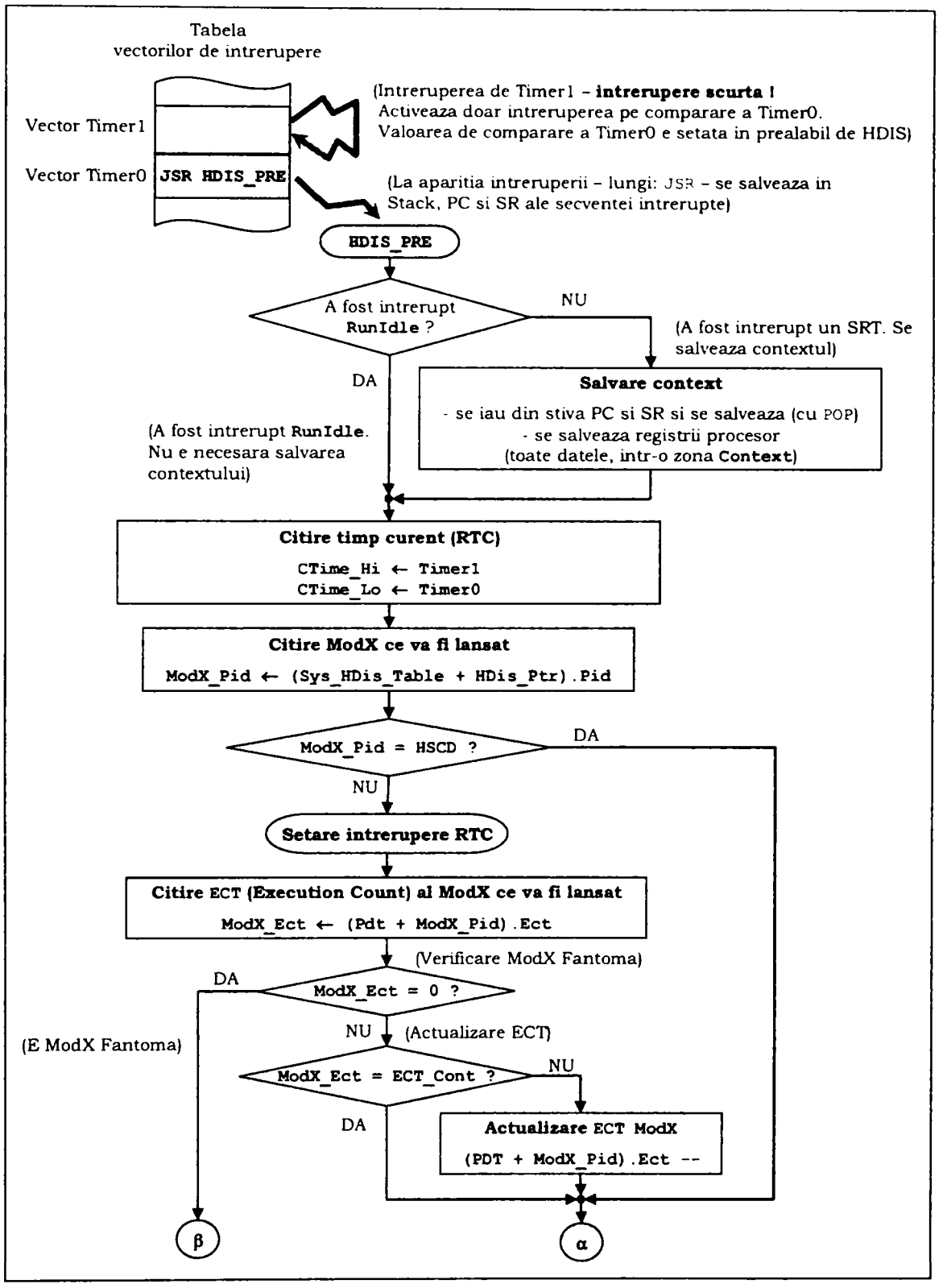

Figura 10-5. Organigrama de operare a componentei HDIS_PRE

Problema a fost rezolvată, în primul rând prin verificarea apariţiei acestor situaţii de depăşire a capacităţii primului timer. Verificarea se face comparând cuvintele cele mai semnificative ale valorilor timpilor de start, pentru ModX-ul curent şi cel următor, din Tabela de Dispatch. Prin această comparare se verifică

practic dacă valoarea din `Timer1` este aceeaşi pentru cei doi timpi, sau nu. Dacă este aceeaşi, înseamnă că timpul ce se va scurge de la execuţia ModX-ului curent şi până la lansarea următorului ModX, este suficient de mic pentru capacitatea de contorizare disponibilă în `Timer0`. Dacă se confirmă depăşirea capacităţii curente a `Timer0`, se vor programa regiştrii de comparare ai ambelor timere, cu valorile corespunzătoare momentului de start al ModX-ului următor, şi se va activa doar întreruperea pentru `Timer1`.

Rutina de tratare a întreruperii generate de `Timer1` are doar rolul de activare a întreruperii pentru `Timer0`. În arhitectura DSP56307, ea este denumită "întrerupere scurtă", pentru că nu implică salvarea stării procesor şi a contorului de program în stivă.

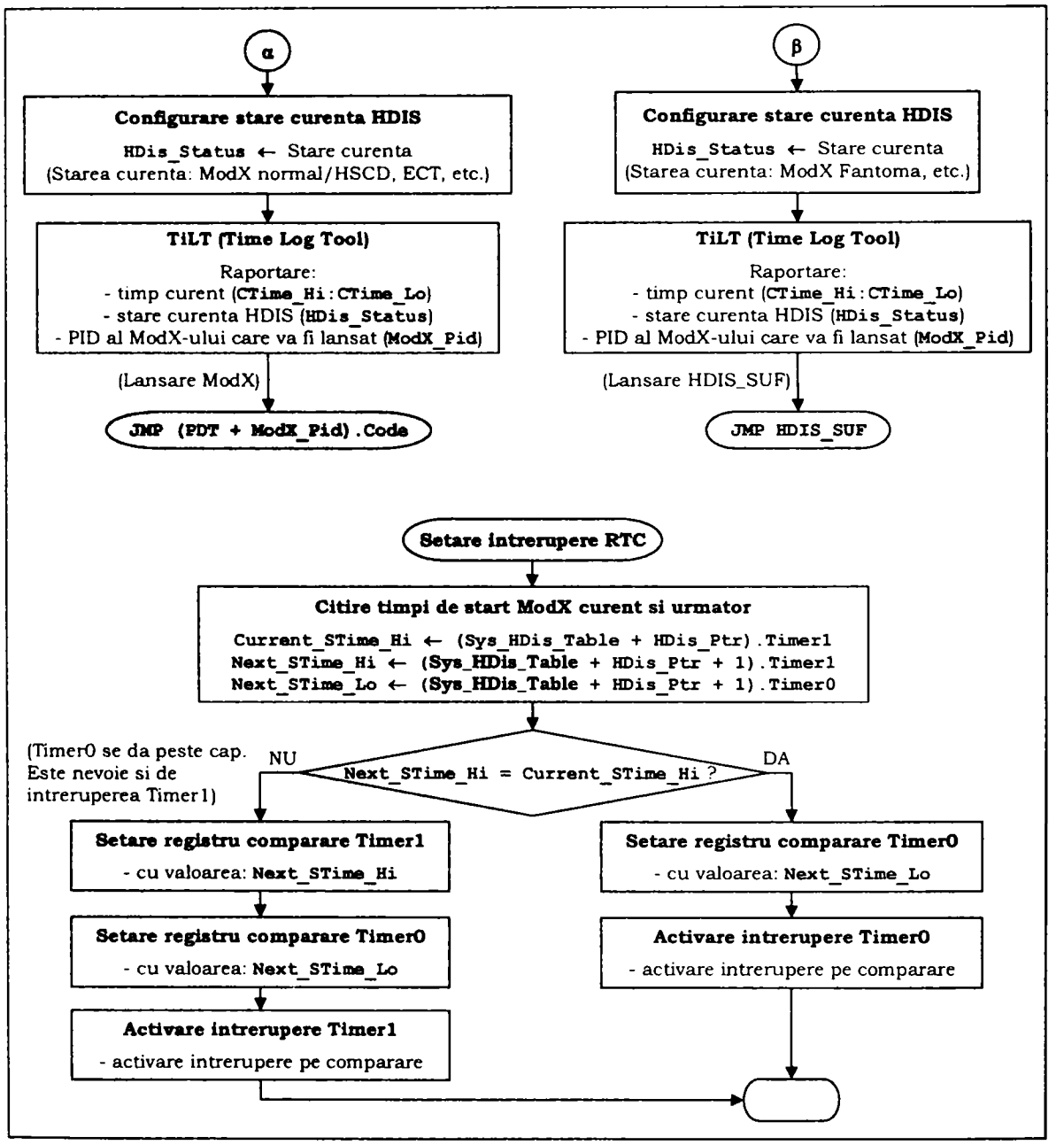

Figura 10-6. Organigrama HDIS_PRE (continuare)

După apelarea lui HDIS_PRE de către întreruperea generată de `Timer0`, se verifică prima dată dacă a fost întreruptă execuţia unui task lejer, caz în care este necesară salvarea contextului său de execuţie într-o zonă de memorie sistem dedicată, denumită "Context". Operaţia de verificare se bazează pe informaţia de stare pe care o menţine HDIS de la o execuţie la alta, prin intermediul unei variabile proprii (`HDis_Status`).

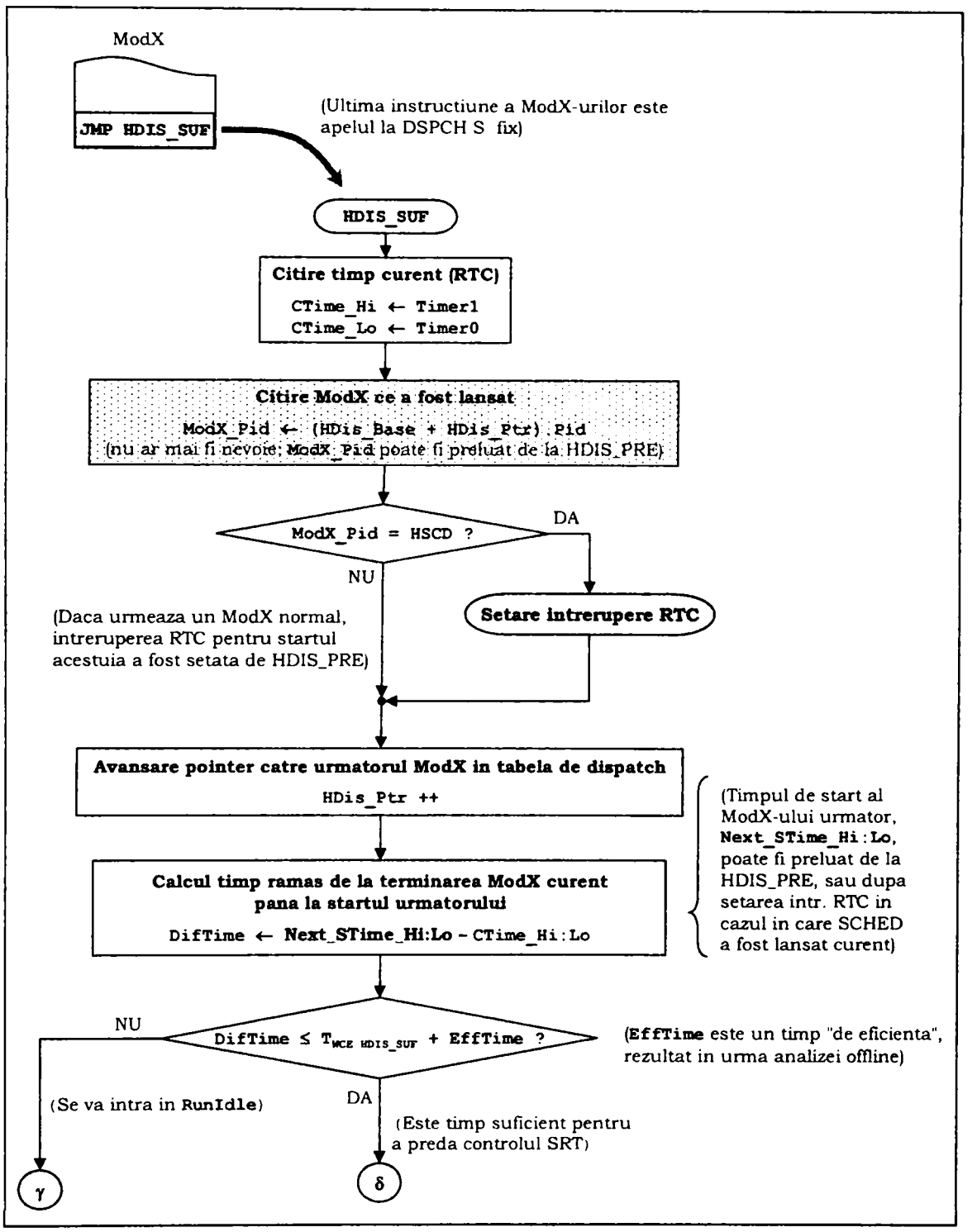

Figura 10-7. Organigrama de operare a componentei HDIS_SUF

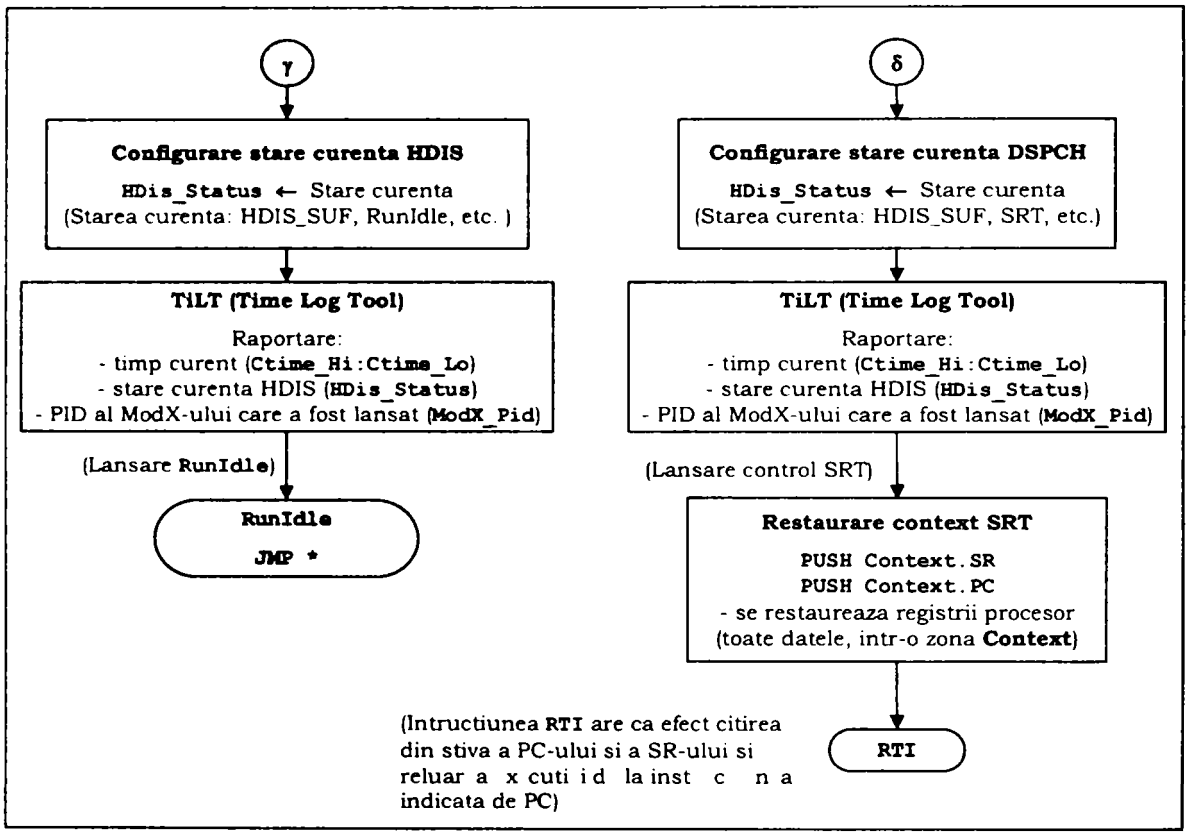

Figura 10-8. Organigrama HDIS_SUF (continuare)

Figura 10-7, continuată cu Figura 10-8, prezintă organigrama de operare a componentei sufix a executivului. HDIS_SUF este apelată de către ModX-ul care îşi încheie execuţia în contextul strict al nucleului.

Se poate observa din organigramele celor două componente ale HDIS că, de fiecare dată când sunt activate, are loc citirea şi înregistrarea momentului curent de timp (se citesc contoarele de timer ale ceasului timp-real). Informaţia de timp este utilizată în rapoartele pe care HDIS le generează prin intermediul subrutinei TiLT.

Subrutina TiLT gestionează o tabelă (un buffer circular), în care înscrie, pe lângă momentul curent al apariţiei unui eveniment, informaţii legate de tipul evenimentului, starea curentă a HDIS şi contorul de execuţii al ModX-ului care a fost procesat de HDIS.

10.4 Planificatorul HRT al HARETICK: HSCD

Planificatorul contextului strict de execuţie al nucleului HARETICK este un modul esenţial pentru operarea corectă a sistemului. Proprietăţile şi parametrii HSCD caracterizează şi influenţează în mod decisiv comportarea şi operarea STR în ceea ce priveşte task-urile cu specificaţii temporale stricte.

Task-ul HSCD implementează algoritmii de planificare non-preemptivă introduşi şi studiaţi în Capitolele 5 şi 6, asupra seturilor de task-uri TR stricte – ModX-urile (Capitolul 4). În cele ce urmează vom prezenta principiile de proiectare şi implementare, vom descrie funcţionarea şi vom analiza principalii parametri ai planificatorului HRT al nucleului HARETICK. Aşa cum se va vedea, task-ul HSCD implementează o serie de algoritmi şi operaţii complexe, care se reflectă în durata sa de execuţie – considerabil mai mare decât marea majoritate a celorlaltor task-uri din sistem. În consecinţă, proiectarea şi implementarea HSCD trebuie tratată cu o deosebită atenţie, astfel încât performanţa şi eficienţa de operare ale sistemului să fie cât mai puţin afectate de execuţia acestui task, care este complet transparent pentru orice aplicaţie din sistem.

10.4.1 Principii de proiectare, implementare şi operare ale HSCD

Obiectivul final în ceea ce priveşte planificarea task-urilor stricte din cadrul unui sistem (aplicaţii) timp-real îl reprezintă implementarea cât mai eficientă, pe platforma HARETICK, a algoritmilor care să rezolve planificarea non-preemptivă a unui set fezabil de ModX-uri cu dependenţe de program, care să includă şi task-uri cu execuţie fixă în cadrul perioadei (FModX-uri). Evidenţiem două aspecte generale legate de implementarea HSCD:

(a) Eficienţa: execuţia task-ului de planificare trebuie să se desfăşoare de fiecare dată într-un interval de timp cât mai scurt, pentru ca impactul asupra eficienţei întregului sistem să fie cât mai mic;

(b) Fezabilitatea: seturile de task-uri (ModX-uri) care vor fi planificate pe parcursul operării sistemului sunt în prealabil analizate offline din punct de vedere al fezabilităţii, cu ajutorul mediului de dezvoltare INVERTA (descris în Capitolul 7), prin aplicarea algoritmilor de planificare similari cu cei pe care îi implementează şi HSCD.

În stadiul curent al cercetării şi dezvoltării, nucleul HARETICK dispune de o versiune preliminară, dar complet funcţională de HSCD, capabil să rezolve planificarea seturilor de ModX-uri independente, cu execuţie fixă în cadrul perioadei (FModX-uri). *Însuşi task-ul de planificare, HSCD, este la rândul său un FModX*, de unde rezultă o serie de proprietăţi importante ale planificării pe platforma ţintă:

- Ciclurile de planificare sunt periodice în timp;
- Numărul de planificări de task-uri (FModX-uri) variază de la un ciclu de planificare la altul;
- Tabela de Dispatch a nucleului conţine un număr variabil de înregistrări de la un ciclu de planificare la altul, impunându-se totuşi o limită superioară (definită de dimensiunea totală a tabelei).

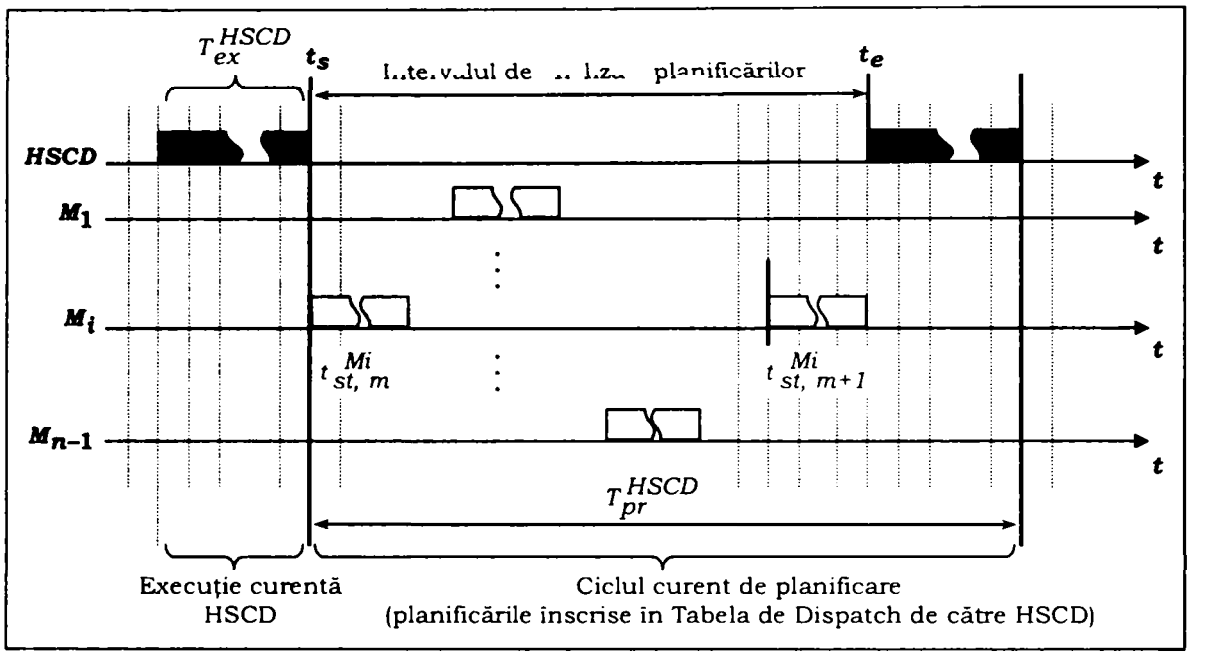

Figura 10-9. Structura unui ciclu de planificare cu HSCD

În Figura 10-9 este prezentat un ciclu de planificare (ciclul curent), pentru un set $\boldsymbol{M} = \{M_1, M_2, \ldots, M_{n-1}\}$ de FModX-uri, în condiţiile operării planificatorului HSCD – definit la rândul lui ca FModX. Ciclul de planificare este definit în cadrul execuţiei curente a HSCD, fiind delimitat de instanţele temporale corespunzătoare terminării execuţiei curente (t_s) şi a celei următoare ale lui HSCD. Durata ciclurilor de planificare este egală cu perioada lui HSCD.

Cum fiecare ciclu se termină cu planificarea unei execuţii HSCD, algoritmii implementaţi vor analiza şi completa înregistrările Tabelei de Dispatch cu planificările FModX-urilor din setul $\boldsymbol{M}$, pentru intervalul:

$$\left[t_s,\ t_e = t_s + T_{pr}^{HSCD} - T_{ex}^{HSCD}\right) \tag{10-1}$$

Algoritmul de planificare online, non-preemptivă, a setului $\boldsymbol{M} \cup \{HSCD\}$ de FModX-uri se bazează pe următoarele elemente:

- Analiza fezabilităţii a fost realizată în prealabil (înaintea încărcării aplicaţiei corespunzătoare pe platforma ţintă), prin aplicarea algoritmului similar cu cel pe care îl va implementa HSCD (vezi Capitolul 5);
- În urma analizei de fezabilitate, pentru fiecare FModX din set, rezultă valoarea parametrului $T_{st}^{M_i}$ (intervalul de start al lui M_i în cadrul perioadei), care este de regulă direct proporţional cu perioada FModX-ului;
- Două execuţii consecutive ale oricărui FModX sunt despărţite întotdeauna de intervalul de timp egal cu perioada acestuia, $T_{pr}^{M_i}$ (vezi şi Figura 10-10);
- Prima execuţie a fiecărui FModX se planifică la momentul $t_{st,1}^{M_i} = T_{st}^{M_i}$, egal cu intervalul de start al FModX-ului în cadrul perioadei sale.

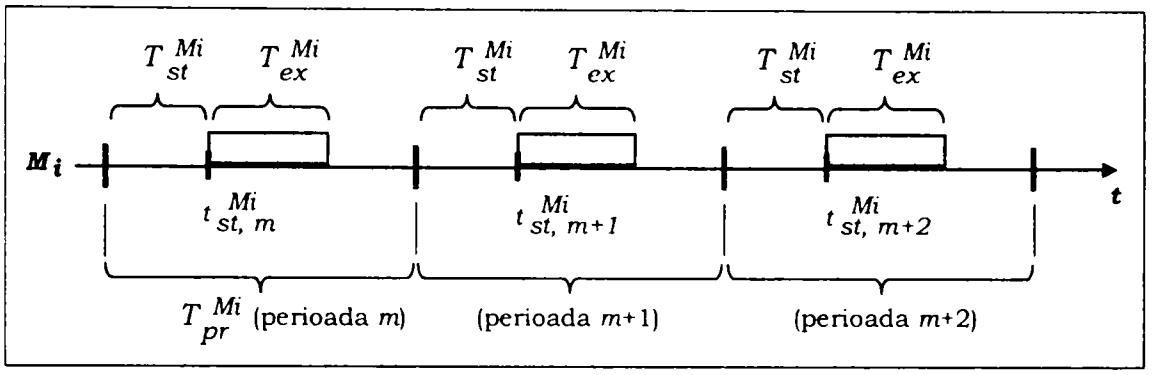

Figura 10-10. Execuţia unui FModX pentru mai multe perioade

Având în vedere considerentele enumerate anterior, s-a ales un mod de operare al planificatorului online HSCD, bazat pe două faze: faza de *pre-planificare*, pentru completarea Tabelei de Dispatch cu planificările din primul ciclu, şi faza de planificare normală, ce urmează pre-planificării, pe toată durata operării sistemului.

Algoritmul pe care îl implementează HSCD se bazează pe următorii parametri ai FModX-urilor din set: perioada ($T_{pr}^{M_i}$ ≡ `PER_Hi:Lo`), durata de execuţie (timpul maxim de execuţie, WCET: $T_{ex}^{M_i}$ ≡ `WCE`), intervalul de start ($T_{st}^{M_i}$ ≡ `DLY_Hi:Lo`). Elementele din paranteze aparţin înregistrării corespunzătoare FModX-ului M_i din cadrul tabelei cu descriptorii de proces, PDT (vezi Figura 9-6 şi Tabelul 9-1 din Secţiunea 9.4.1.)

Operarea de principiu a algoritmului de planificare online implementat de HSCD cuprinde următorii paşi principali:

0) În faza de pre-planificare, se completează Tabela de Dispatch (HDis_Tab) a sistemului cu planificările FModX-urilor din setul dat, începând cu prima perioadă de execuţie a fiecăruia. Primul ciclu se încheie atunci când s-a ajuns să se planifice o execuţie HSCD.

Momentul iniţial al planificării se consideră $t_{S0} = 0$, corespunzător activării contextului strict de execuţie al HARETICK. Instanţele de start ale primelor execuţii ale ModX-urilor planificate în primul ciclu sunt egale chiar cu intervalele de start corespunzătoare ($T_{st}^{M_i}$ ≡ `DLY_Hi:Lo`).

După fiecare planificare reuşită a unui FModX, se actualizează parametrul `SCT_Hi:Lo` al acestuia din tabela PDT cu momentul de timp corespunzător planificării. Astfel, `SCT_Hi:Lo` memorează ultimul moment al unei planificări reuşite a FModX-ului respectiv.

În momentul planificării execuţiei HSCD care termină primul ciclu (t_e, vezi şi Figura 10-9), parametrul `SCT_Hi:Lo` corespunzător din PDT va fi actualizat cu instanţa de timp a execuţiei următoare celei planificate (deci cu instanţa execuţiei HSCD care va termina ciclul următor):

$$\texttt{SCT\_Hi:Lo}_{HSCD} \leftarrow t_e + \texttt{PER\_Hi:Lo}_{HSCD} \qquad (10\text{-}2)$$

1) Înainte de a începe calculul planificărilor ciclului curent, HSCD preia instanța de sfârşit a intervalului de analiză, t_e (conform (10-1)), din parametrul `SCT_Hi:Lo` corespunzător din PDT, a cărui valoare a fost pregătită de execuția anterioară a HSCD:

$$t_e \leftarrow \texttt{SCT\_Hi:Lo}_{HSCD} \tag{10-3}$$

2) În cadrul unei bucle, HSCD testează, pentru fiecare FModX din setul $\boldsymbol{M}$, dacă poate fi planificat în intervalul de analiză, cu alte cuvinte, dacă pentru fiecare $M_i \in \boldsymbol{M}$, $i = 1 \,..\, n-1$, se verifică relația:

$$t_{Si} = \texttt{SCT\_Hi:Lo}_{Mi} + \texttt{PER\_Hi:Lo}_{Mi} < t_e \tag{10-4}$$

În $\texttt{SCT\_Hi:Lo}_{Mi}$ este memorată ultima instanță de planificare reuşită a FModX-ului M_i.

2.1) Dacă (10-4) nu se verifică, înseamnă că FModX-ul M_i nu mai poate fi planificat în ciclul curent, şi se trece la următorul FModX din set (M_{i+1}).

2.2) Dacă se verifică (10-4), parametrul $\texttt{SCT\_Hi:Lo}_{Mi}$ din PDT este actualizat cu valoarea t_{Si}, şi perechea $\{i, t_{Si}\}$ (adică PID-ul lui M_i şi instanța de planificare) este inserată într-o listă de planificări ale ciclului curent, ordonată după timp.

3) În momentul în care s-a terminat baleierea tuturor FModX-urilor din $\boldsymbol{M}$ şi s-a construit lista ordonată după timp a planificărilor acestora în cadrul ciclului curent, HSCD completează Tabela de Dispatch.

HDis_Tab este completată, pornind de la prima înregistrare a sa (de la baza tabelei), parcurgând lista în ordinea crescătoare a instanțelor de planificare. Ca rezultat, HDis_Tab va conține toate perechile de forma $\{\text{PID}_i, t_{Si}\}$ cu planificările FModX-urilor din $\boldsymbol{M}$ pentru ciclul curent.

4) În final, HSCD completează următoarea înregistrare liberă din HDis_Tab cu planificarea HSCD care încheie ciclul curent: $\{\text{PID}_{HSCD}, t_e\}$, şi îşi actualizează valoarea parametrului $\texttt{SCT\_Hi:Lo}_{HSCD}$ din PDT, conform (10-2).

10.4.2 Analiza codului HSCD

Aşa cum s-a subliniat în paragrafele anterioare, implementarea algoritmului de planificare online trebuie făcută cu o deosebită atenție, avându-se în vedere în special minimizarea timpului de execuție al HSCD.

În forma actuală, HSCD a fost implementat pe platforma Motorola DSP56307, în întregime în limbaj de asamblare. Timpul maxim de execuție al HSCD depinde în mod direct de următoarele structuri de cod:

(i) Secvența inițializărilor (include pasul 1 din algoritmul descris mai sus);
(ii) Bucla de analiză a planificărilor pentru ciclul curent (include pasul 2). Secvența de analiză este practic formată din două bucle imbricate.

Bucla exterioară testează planificabilitatea în cadrul ciclului curent a fiecărui FModX, pe rând. În consecință, durata de execuție a buclei exterioare depinde de numărul total de FModX-uri din setul ***M*** (excluzând pe HSCD), pe care îl notăm cu n, şi de numărul maxim de planificări admise pentru ciclurile de planificare, cu alte cuvinte, de dimensiunea utilă maximă a Tabelei de Dispatch a sistemului (totalul de înregistrări din HDis_Tab, excluzând planificarea HSCD de la finalul ciclului), notată aici cu λ. În consecință, timpul maxim de execuție a buclei exterioare este de ordinul:

$$WCET_{BuclaExt} = \boldsymbol{O}(n + \lambda) \quad (10\text{-}5)$$

Astfel, în (10-5), λ reprezintă totalul încercărilor reuşite de planificare în cadrul ciclului curent (trebuie să fie mai mic sau egal cu numărul maxim de planificări admise de HDis_Tab), iar n reprezintă totalul încercărilor nereuşite de planificare (câte o încercare pentru fiecare FModX din ***M***).

Bucla interioară se parcurge de fiecare dată când o încercare de planificare a reuşit. Execuția buclei interioare constă din parcurgerea listei ordonate a planificărilor din ciclul curent şi inserarea noii planificări în listă. Cum dimensiunea curentă a listei creşte cu fiecare nouă planificare reuşită, şi cum λ este numărul maxim de planificări posibile într-un ciclu, durata maximă de execuție a buclei interioare este de ordinul:

$$WCET_{BuclaInt} = \boldsymbol{O}(1 + 2 + \ldots + \lambda) = \boldsymbol{O}(\lambda(\lambda+1)/2) \quad (10\text{-}6)$$

(iii) Bucla de parcurgere a listei ordonate pentru citirea planificărilor şi completarea Tabelei de Dispatch. Secvența de cod include operațiile specificate la pasul 3 al algoritmului HSCD. Durata maximă de execuție este de ordinul:

$$WCET_{CitireListă} = \boldsymbol{O}(\lambda) \quad (10\text{-}7)$$

(iv) Secvența finală de cod a HSCD, ce include pasul 4 al algoritmului.

Figura 10-11 prezintă analiza duratei maxime de execuție a secvențelor de cod corespunzătoare buclelor imbricate de determinare a planificărilor pentru ciclul curent (ii). Timpii de execuție ai secvențelor de cod sunt calculați în cicluri de tact procesor (CLK). Pentru durata maximă de execuție a secvenței de determinare a planificărilor a rezultat valoarea:

$$WCET_{AnalizăPlanificări} = 15\cdot(\lambda + n) + 2\cdot n + 48\cdot\lambda + 13\cdot\lambda\cdot(\lambda + 1) \text{ [CLK]} \quad (10\text{-}8)$$

Analiza întregii secvențe de cod a planificatorului online HSCD este ilustrată în Figura 10-12. Durata maximă de execuție a codului HSCD devine:

$$WCET_{CodHSCD} = 56 + 71\cdot\lambda + 15\cdot(\lambda + n) + 2\cdot n + 13\cdot\lambda\cdot(\lambda + 1) \text{ [CLK]} \quad (10\text{-}9)$$

Observație

Datorită faptului că HSCD este, la rândul său, un FModX care se execută în contextul HRT al nucleului HARETICK, la durata sa de execuție se adaugă şi cea a executivului HDIS (Secțiunea 9.6):

$$WCET_{HSCD} = WCET_{CodHSCD} + WCET_{HDIS}$$

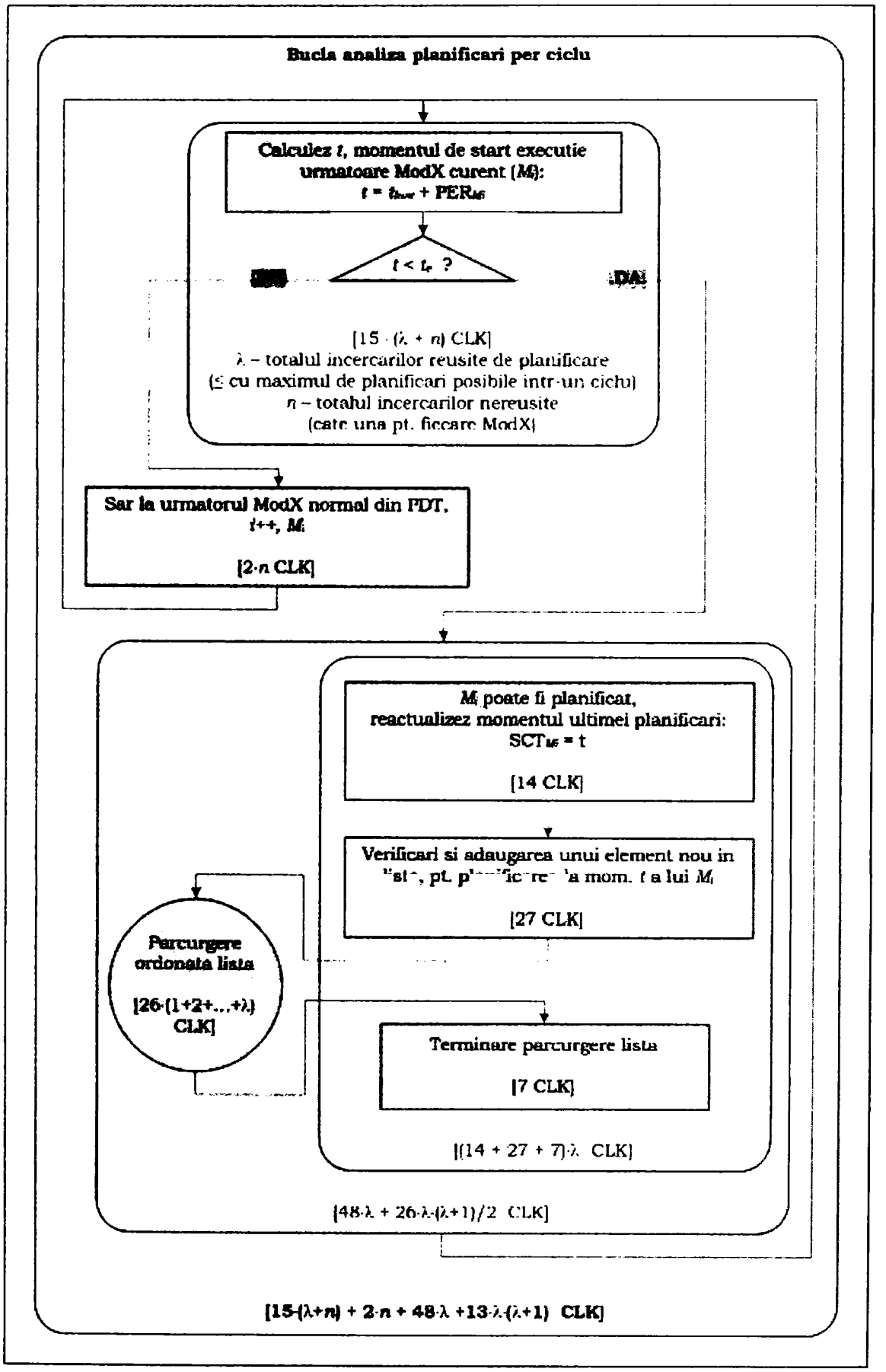

Figura 10-11. Secvenţa de cod HSCD pentru analiza planificărilor din ciclul curent

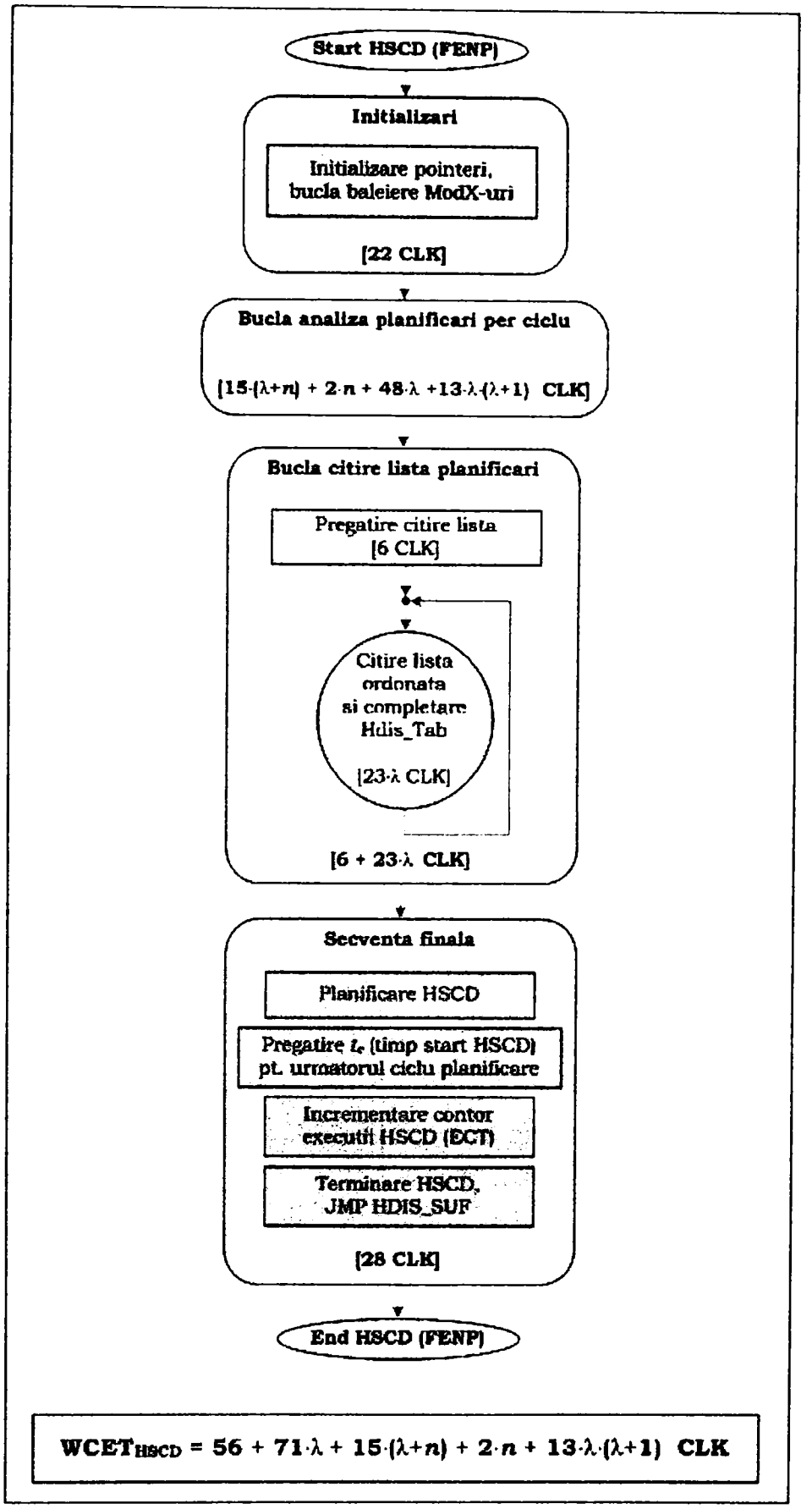

Figura 10-12. Structura întregii secvenţe de cod a HSCD

10.5 Concluzii

Nucleul HARETICK este conceput pentru a oferi suport execuţiei pe platforme de control încorporat pentru aplicaţiile timp-real, care, de regulă, conţin atât task-uri cu specificaţii temporale stricte (task-uri HRT) cât şi task-uri cu specificaţii lejere (SRT). În HARETICK, cele două tipuri de task-uri se execută în cadrul a două contexte concurente:

(1) Contextul strict de execuţie (contextul HRT): oferă suportul necesar execuţiei task-urilor cu specificaţii temporale stricte, modelate cu ajutorul ModX-ului, garantând respectarea termenelor impuse la specificare.

ModX-urile sistem şi ale aplicaţiei sunt planificate cu ajutorul unui task sistem (ModX, la rândul său), HSCD, şi se execută în regim non-preemptiv.

Contextul HRT are prioritatea cea mai mare în cadrul nucleului şi este activat într-un mod complet predictibil, la instanţele de timp ce marchează începutul execuţiei ModX-urilor planificate. Activarea contextului HRT este realizată de către executivul nucleului, HDIS.

(2) Contextul lejer de execuţie (contextul SRT): este specific task-urilor cu specificaţii lejere de comportare temporală. Contextul SRT a fost propus şi dezvoltat în cadrul nucleului HARETICK din considerente legate de creşterea eficienţei şi flexibilităţii întregului sistem.

Contextul SRT are precedenţă mai mică faţă de cel HRT, fiind întrerupt de către acesta prin mecanismul de ceas timp-real (RTC) al sistemului. Comutarea celor două contexte este efectuată de către HDIS, care, în aceste situaţii se ocupă şi cu salvarea şi restaurarea contextului SRT.

Planificarea task-urilor lejere este realizată într-o manieră clasică, de tip "time-sharing", pe bază de priorităţi, de către SSCD – planificatorul contextului SRT (un task SRT, la rândul său).

Datorită importanţei task-urilor HRT în sistemele timp-real, preocupările noastre de cercetare-dezvoltare s-au concentrat asupra modului în care nucleul HARETICK tratează aceste task-uri în cadrul contextului strict de execuţie. Pe parcursul operării, ModX-urile se pot găsi în 5 stări distincte:

- NOP ("No Operation") descrie starea ModX-urilor care nu au fost încărcate încă pe platforma ţintă şi nucleul nu are cunoştinţă de existenţa acestora;
- RDY ("Ready for Scheduling") este starea caracteristică ModX-urilor încărcate în sistemul HARETICK. Structurile de date necesare reprezentării ModX-urilor în cadrul nucleului au fost alocate şi iniţializate cu valorile corespunzătoare;
- SCD ("Scheduled") descrie starea ModX-urilor care sunt planificate în cadrul ciclului curent de planificare, fiind astfel prezente în Tabela de Dispatch a nucleului;
- RUN ("Running") reprezintă starea unui singur ModX, care la un moment dat este în execuţie în cadrul sistemului ţintă;

- GST ("Ghost") descrie starea ModX-ului planificat pentru execuţie la momentul curent, dar care are contorul de execuţii nul. Execuţia efectivă a ModX-ului nu se mai petrece.

Rolul de a comuta cele două contexte de execuţie ale HARETICK şi de a salva şi restaura informaţia contextului SRT la întreruperea acestuia, îi revine componentei de bază a nucleului – executivul HDIS. Executivul este compus din trei module: prefix (HDIS_PRE), sufix (HDIS_SUF) şi TiLT ("Time Log Tool") şi utilizează o structură sistem statică pentru lansarea în execuţie a ModX-urilor planificate în cadrul unui ciclu: Tabela de Dispatch (HDis_Tab).

HDIS_PRE este apelat întotdeauna de către întreruperea ceasului timp-real (RTC) a sistemului, la instanţele de timp corespunzătoare planificării pentru execuţie a ModX-ului curent din Tabela de Dispatch. Înaintea lansării în execuţie a ModX-ului curent, HDIS_PRE setează regiştrii de comparare ai RTC pentru generarea întreruperii ce va iniţia execuţia ModX-ului următor din HDis_Tab. De asemenea, dacă a fost întrerupt contextul SRT, componenta prefix a executivului salvează starea şi regiştrii procesor.

HDIS_SUF este apelat la terminarea execuţiei fiecărui ModX din sistem şi are rolul de a decide comutarea contextelor (HRT către SRT). Efectuarea operaţiei de comutare este precedată de restaurarea contextului task-ului SRT întrerupt de către HDIS_PRE.

Rutina TiLT are rolul de a înregistra principalele evenimente legate de execuţia ModX-urilor într-o structură de tip buffer circular.

Planificarea execuţiei în contextul strict al HARETICK este efectuată de către HSCD, pe baza noţiunii de "ciclu de planificare", care este definit ca intervalul de timp dintre două execuţii consecutive ale HSCD.

În prezent, a fost proiectată, implementată şi testată pe platforma Motorola DSP56307 o versiune de HSCD bazată pe algoritmul de planificare non-preemptivă a ModX-urilor cu execuţie fixă în cadrul perioadei (FModX-uri): algoritmul FENP (vezi Secţiunea 5.4). Proiectarea şi implementarea planificatorului (scris în limbaj de asamblare pentru platforma ţină menţionată) au fost realizate cu o deosebită atenţie, vizându-se minimizarea duratei de execuţie, astfel încât eficienţa sistemului să fie cât mai puţin afectată.

11 Testarea şi evaluarea performanţelor sistemului HARETICK

În stadiul curent de dezvoltare şi implementare, nucleul HARETICK oferă suport complet funcţional pentru aplicaţiile critice compuse din ModX-uri cu execuţie fixă în cadrul perioadei (FModX-uri). Astfel, modulele sistem de bază ale contextului strict din cadrul nucleului au fost implementate şi testate cu succes:

- structurile de date sistem de bază (tabela descriptorilor de proces PDT, Tabela de Dispatch HDis_Tab, etc.),
- ceasul timp-real (RTC), bazat pe un timer procesor pentru partea cea mai puţin semnificativă şi o variabilă sistem pentru partea cea mai semnificativă,
- task-ul de iniţializare SYSINIT,
- o versiune simplă de LOADER,
- executivul HDIS,
- diferite versiuni ale planificatorului HSCD.

Capitolul de faţă prezintă rezultatele aplicaţiilor şi testelor de evaluare a performanţelor componentelor nucleului şi ale întregului sistem. Experimentele se bazează pe implementarea HARETICK pe platforma Motorola DSP56307 EVM [Motorola 99].

Platforma ţintă a fost configurată pentru operarea la o frecvenţă a tactului intern de 32 MHz (1 CLK), iar ceasul timp-real al HARETICK aplică un factor de divizare de 1/4 frecvenţei de tact procesor. Prin urmare, unitatea de timp sistem are valoarea (vezi şi Subcapitolul 9.1):

$$\Delta t_{RTC} = 4 \cdot \Delta t_{CLK} = 125 \text{ ns } (8 \text{ MHz}) \qquad (11\text{-}1)$$

11.1 Testarea şi analiza operării executivului HDIS

Executivul HDIS este modulul de bază al nucleului HARETICK, fiind responsabil de operarea corectă a contextului strict de execuţie (cu implicaţii directe asupra mediului şi/sau a operatorilor în cazul aplicaţiilor critice). De asemenea, HDIS este responsabil de comutarea corectă a celor două contexte de execuţie ce operează concurent în cadrul nucleului: contextul HRT şi SRT.

Implementarea şi testarea executivului HDIS este unul din primii paşi necesari pentru dezvoltarea nucleului HARETICK. În acest scop, pe platforma Motorola DSP56307 EVM, a fost conceput şi implementat un cadru simplu de operare, în care o structură particulară a Tabelei de Dispatch (HDis_Tab) şi o versiune simplă de planificare HRT sunt elementele cheie. În acest mediu de operare, a fost apoi dezvoltată o aplicaţie simplă, cu ajutorul căreia se pot efectua teste şi măsurători practice ale execuţiei în contextul strict al nucleului.

Structura HDis_Tab a fost concepută ca un buffer circular cu dimensiune bine stabilită (1000 înregistrări pentru planificări de ModX-uri normale + 1 înregistrare pentru planificarea HSCD). Prima înregistrare din HDis_Tab este alocată totdeauna planificărilor HSCD. În Figura 11-1 este prezentată structura HDis_Tab şi configuraţia sa pentru cazul primului ciclu de planificare a ModX-urilor aplicaţiei de testare a executivului.

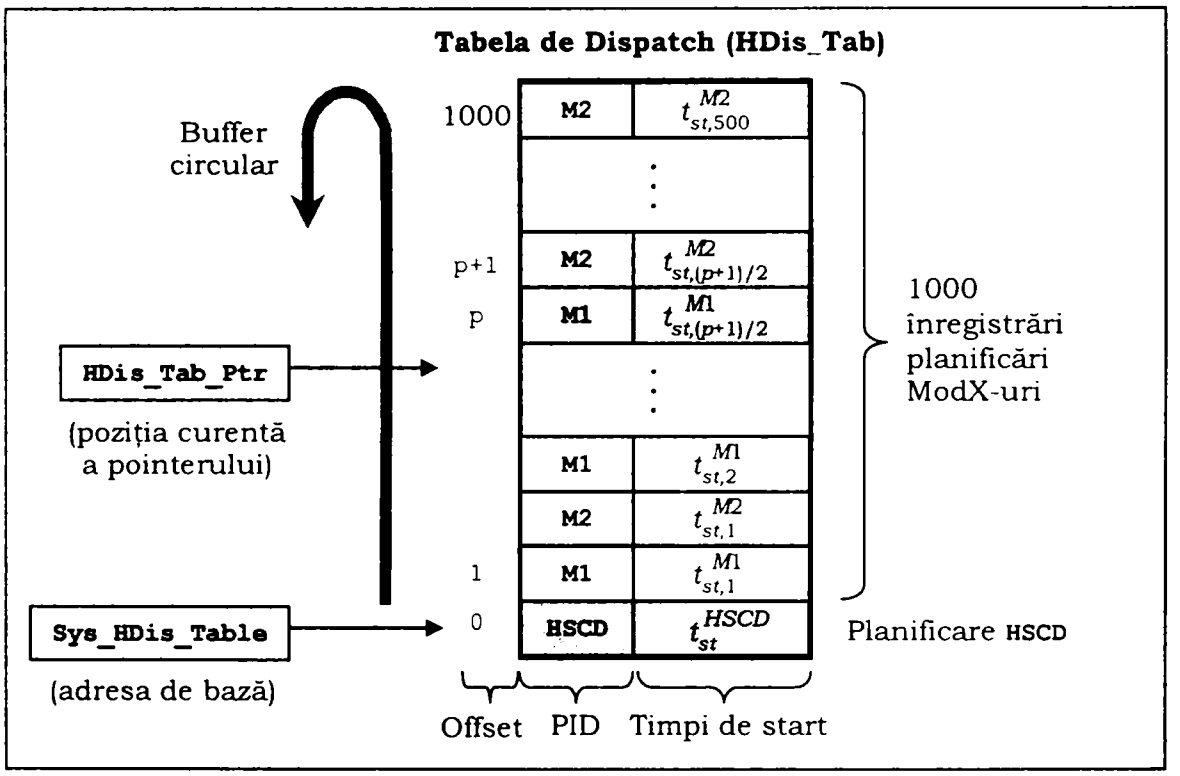

Figura 11-1. Structura HDis_Tab ca buffer circular, pentru planificarea simplă

Modulul de planificare HSCD (un ModX, rândul său), implementează un algoritm simplu, ce constă din completarea alternativă a înregistrărilor HDis_Tab cu planificările ModX-urilor aplicaţiei (vezi Figura 11-1). Timpii de planificare ai ModX-urilor sunt calculaţi în aşa fel încât momentul curent al planificării unui anume ModX apare la terminarea execuţiei ModX-ului planificat anterior. De exemplu, pentru prima planificare a lui M_2 din Figura 11-1 (înregistrarea a treia din HDis_Tab), momentul de start este calculat astfel:

$$t_{st,1}^{M_2} = t_{st,1}^{M_1} + t_{ex}^{M_1} \tag{11-2}$$

Când execuţia în contextul strict al nucleului a ajuns la ultima planificare din HDis_Tab, pointerul `HDis_Tab_Ptr` va fi actualizat automat pentru a referi prima înregistrare a tabelei, care reprezintă planificarea HSCD ce defineşte un nou ciclu de planificare.

Principalii parametrii de implementare a componentelor nucleului HARETICK pentru procedurile de testare a executivului HDIS, sunt sintetizaţi în Tabelul 11-1.

Tabelul 11-1. Parametrii de implementare a componentelor HARETICK

Denumire	Descriere	Dimensiune [24-bit words]	WCET [CLK]
PDT	Tabela descriptorilor de proces	1300	–
HDis_Tab	Tabela de Dispatch (1000 + 1 înregistrări)	3003	–
SRT_Context	Zonă de memorie pentru salvarea şi restaurarea contextului SRT	39	–
TiLT_Tab	Buffer pentru rapoartele TiLT	5120	–
Sys_Heap	Zonă de tip "heap" pentru variabilele şi parametrii ModX-ului în execuţie	4096	–
RTC_OVR	Rutina de tratare a întreruperii de depăşire a timerului RTC	26	37
TiLT	Subrutina "Time Log Tool"	33	43
HDIS_PRE	Componenta prefix a executivului	214	226
HDIS_SUF	Componenta sufix a executivului	177	137
HDIS	Executivul nucleului	450	486
HSCD	Planificatorul contextului HRT	75	36534
LOADER	Task-ul de încărcare a aplicaţiilor în sistem şi de gestionare a memoriei (task SRT)	40	–
SYSINIT	Task-ul de iniţializare a sistemului (task SRT)	68	–

Durata maximă de execuţie pentru HDIS ($WCET_{HDIS}$) rezultă din însumarea duratelor maxime ale rutinei de tratare a întreruperii de depăşire (RTC_OVR), a componentei prefix (HDIS_PRE) ce apelează subrutina TiLT, şi a componentei sufix (HDIS_SUF) ce apelează şi ea subrutina TiLT:

$$WCET_{HDIS} = WCET_{RTC\_OVR} + \left(WCET_{HDIS\_PRE} + WCET_{TiLT} \right) +$$

$$+ \left(WCET_{HDIS\_SUF} + WCET_{TiLT} \right)$$

Observaţie

Deoarece la durata de execuţie a oricărui ModX care operează în sistemul HARETICK se adaugă durata executivului, $WCET_{HDIS}$, acest parametru reprezintă o limită minimă. Cu alte cuvinte, pentru orice ModX din sistem nu se va putea considera o durată de execuţie mai scurtă decât $WCET_{HDIS}$ (aici, 486 cicluri de CLK).

De asemenea, comportarea ModX-urilor fantomă este caracterizată de durata de execuţie a HDIS.

Din Tabelul 11-1 se poate observa că planificatorul HSCD are durata de execuţie cea mai mare. Aceasta se datorează numărului relativ mare de înregistrări ale HDis_Tab pe care HSCD trebuie să le completeze (să calculeze planificările): 1000 (ModX-uri normale) + 1 (planificarea execuţiei următoare HSCD).

Aplicaţia dezvoltată cu scopul de a testa, măsura şi evalua performanţele executivului HDIS, este compusă din trei task-uri: task-ul lejer de iniţializare a aplicaţiei, `()AppInit()`, obligatoriu pentru toate aplicaţiile ce rulează pe platforme HARETICK, şi două ModX-uri, M_1 şi M_2. Principalul rol al celor două ModX-uri este comanda unui semnal de ieşire, care să poată fi măsurat cu ajutorul unui osciloscop.

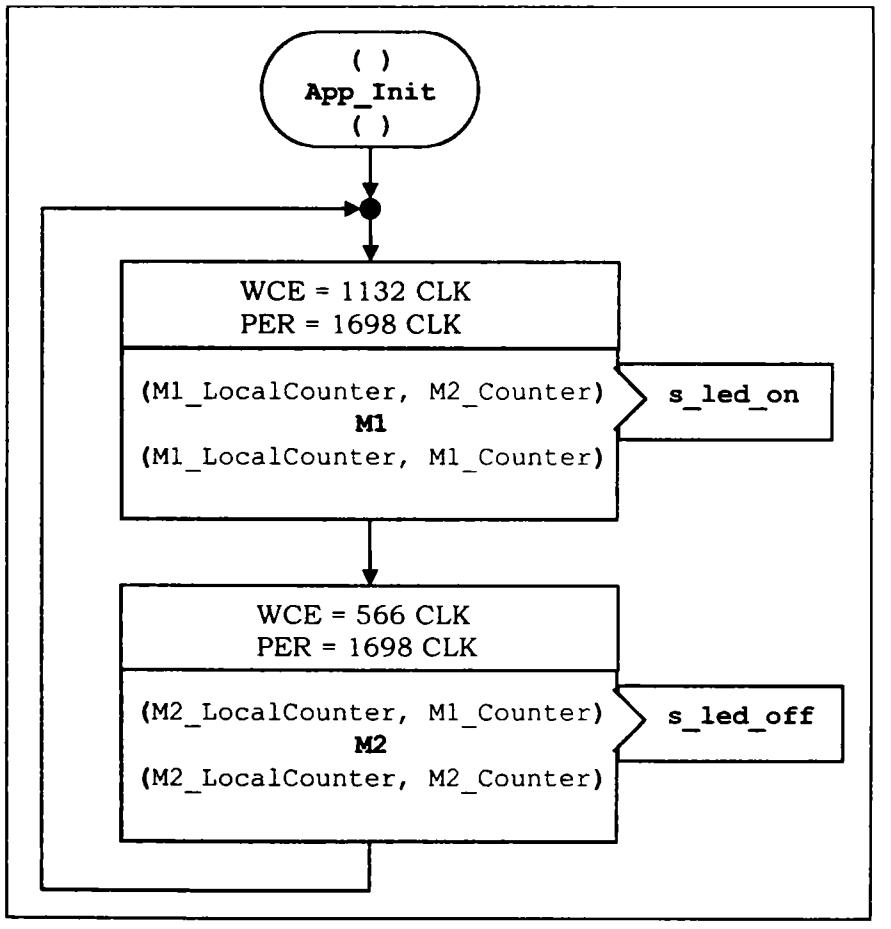

Figura 11-2. Aplicaţia de testare a executivului HARETICK

Figura 11-2 prezintă graful aplicaţiei de testare a executivului. Parametrii de intrare ale task-urilor apar în parantezele superioare iar parametrii de ieşire, în parantezele inferioare denumirii task-urilor.

Din punct de vedere al specificaţiilor funcţionale, ModX-ul M_1 setează un pin de ieşire al platformei ţintă (`s_led_on` ← "1"), după care incrementează contorul local (`M1_LocalCounter` ← `M1_LocalCounter` + 1) şi în final adună valoarea 2 la contorul pe care îl transmite lui M_2 (`M1_Counter` ← `M2_Counter` + 2). Pe de altă parte, ModX-ul M2 resetează pinul de ieşire al sistemului (`s_led_off` ← "0"), îşi incrementează contorul local (`M2_LocalCounter` ← `M2_LocalCounter` + 1), şi adună 2 la contorul pe care îl transmite lui M_1 (`M2_Counter` ← `M1_Counter` + 2).

Observaţie

Înainte de a se executa secvenţele de cod corespunzătoare specificaţiilor funcţionale descrise anterior, fiecare ModX îşi va copia proprii parametri de intrare în variabilele locale corespondente (alocate în zona "Heap" a sistemului).

Se poate observa de exemplu că, în cazul lui M_1, `M1_LocalCounter` este atât un parametru de intrare, cât şi de ieşire, transmiţând astfel o anumită stare (valoare) de la o execuţie la alta a lui M_1.

Terminarea ModX-urilor se face în mod obligatoriu cu apelarea componentei sufix a executivului, HDIS_SUF.

Task-ul de iniţializare al aplicaţiei rulează în context SRT şi are ca principale roluri de a iniţializa parametrii de ieşire ai ModX-urilor M_1 şi M_2 (`M1_LocalCounter` ← 0; `M1_Counter` ← 0; `M2_LocalCounter` ← 0; `M2_Counter` ← 0), şi de iniţializare a portului de ieşire pentru pinul ce va fi comandat de M_1 şi M_2 (`s_led_on` şi `s_led_off`).

Tabelul 11-2. Parametrii de implementare ai celor două ModX-uri ale aplicaţiei

Denumire	Descriere	Dimensiune [24-bit words]	WCET [CLK]
M1	Setează pinul de ieşire (`s_led_on`)	12	1132
M2	Resetează pinul de ieşire (`s_led_off`)	12	566

Aşa cum rezultă din parametrii specificaţi pentru cele două ModX-uri ale aplicaţiei (Tabelul 11-2 şi Figura 11-2) şi din tipul algoritmului de planificare pe care îl implementează HSCD, ModX-urile M_1 şi M_2 vor avea o comportare *pseudo-periodică*, adică periodică pe durata fiecărui ciclu de planificare. Valoarea perioadei ModX-urilor rezultă ca suma duratelor individuale de execuţie a acestora. Comportarea periodică este întreruptă pe parcursul execuţiei efective a planificatorului HSCD.

Ca urmare, la pinul de ieşire al platformei ţintă se va putea măsura un semnal de tip rectangular, cu un factor de umplere de 2/3. Punând în corespondenţă duratele ce caracterizează comportarea aplicaţiei, exprimate în cicli de tact procesor, cu valoarea unităţii de timp sistem, Δt_{RTC} (conform (11-1)), vor rezulta o serie de timpi (Tabelul 11-3) ce pot fi verificaţi practic, prin măsurarea cu ajutorul osciloscopului a semnalului rectangular de ieşire.

Măsurătorile au fost efectuate cu un osciloscop digital de tip HAMEG "Analog and Digital Scope, HM1507-3". Rezultatele obţinute pentru măsurarea perioadei şi a duratei de execuţie a planificatorului HSCD sunt prezentate în Figura 11-3 şi Figura 11-4. Se pot observa execuţiile periodice ale lui M_1 şi M_2 în cadrul unui ciclu de planificare, definit de două execuţii consecutive ale HSCD. Măsurarea duratelor de execuţie ale lui M_1 şi M_2, şi a perioadei acestora, este prezentată în cele trei componente din Figura 11-5.

Tabelul 11-3. Timpii caracteristici ai aplicaţiei (valori rezultate din specificaţii)

Denumire	Descriere	Valoare [μs]
$WCET_{HSCD}$	Durata de execuţie HSCD	1141,750
PER_{HSCD}	Perioada HSCD	27766,750
$WCET_{M1}$	Durata de execuţie M_1	35,500
$WCET_{M2}$	Durata de execuţie M_2	17,750
$PER_{M1} = PER_{M2}$	Perioada M_1 şi M_2	53,250

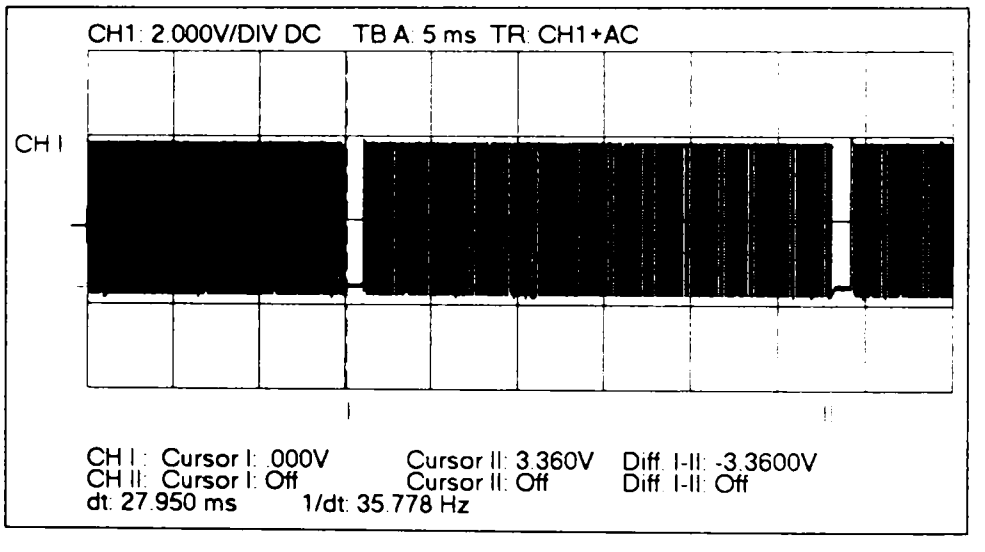

Figura 11-3. Măsurarea perioadei planificatorului HSCD

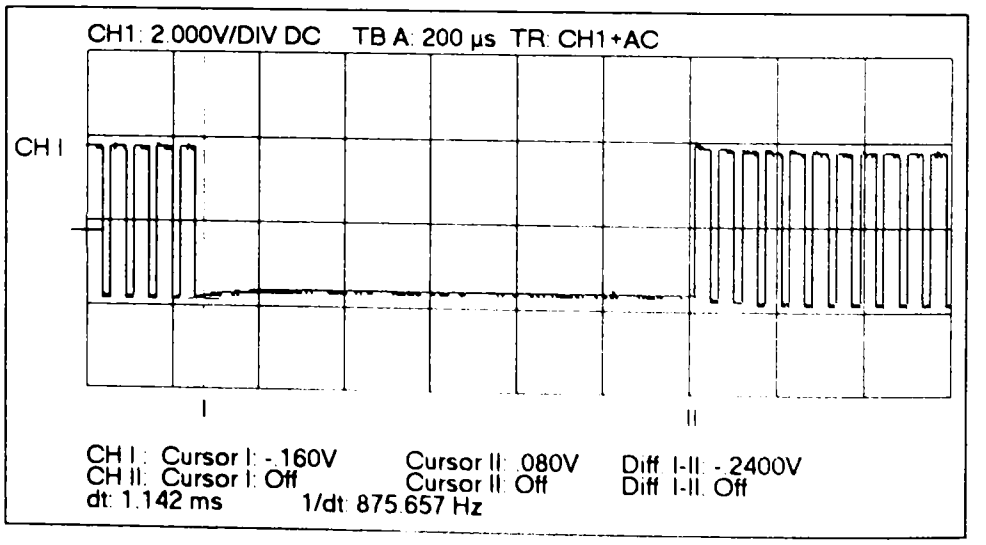

Figura 11-4. Măsurarea duratei de execuţie a HSCD

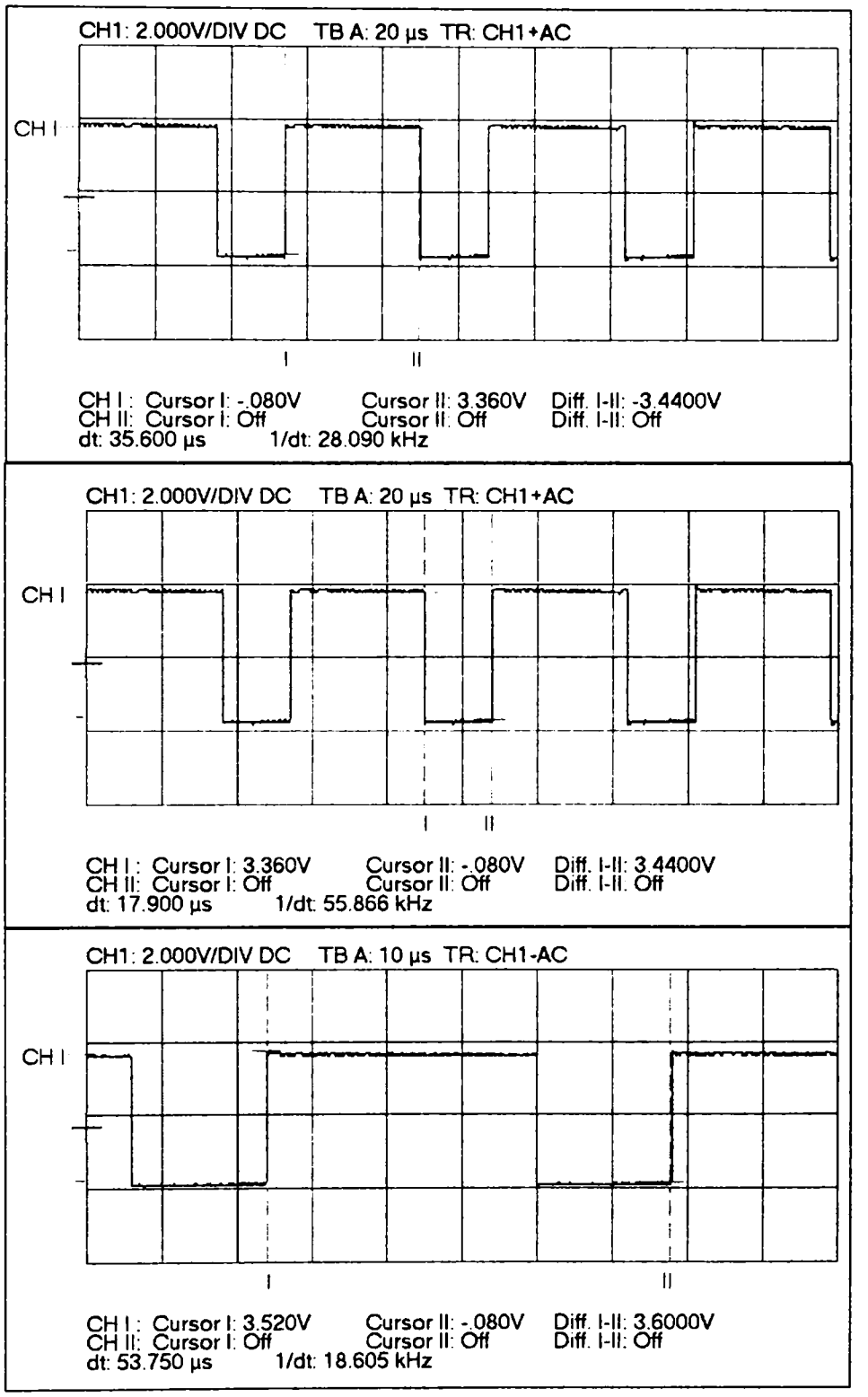

Figura 11-5. Măsurarea duratelor de execuţie a lui M_1 (sus) şi M_2 (mijloc), şi a perioadei celor două ModX-uri (jos)

Rezultatele experimentelor şi măsurătorilor practice confirmă operarea corectă a aplicaţiei, conform specificaţiilor (Tabelul 11-3), şi prin urmare, executivul HDIS al nucleului conferă contextului strict de execuţie al HARETICK caracteristicile dorite şi specificate în faza de modelare şi proiectare a sistemului.

11.2 Aplicaţii pentru evaluarea performanţelor sistemului

Pentru testarea şi evaluarea performanţelor întregului sistem HARETICK, este necesară o configuraţie completă de execuţie a aplicaţiilor timp-real, ce include o versiune funcţională a planificatorului HSCD (spre deosebire de versiunea de planificare simplă utilizată în paragrafele anterioare).

Componentele nucleului HARETICK utilizate pentru aplicaţiile următoare de evaluare a performanţelor sistemului nu sunt modificate faţă de cele descrise anterior (Tabelul 11-1), cu excepţia planificatorului online HSCD.

HSCD implementează o variantă complet funcţională a algoritmului de planificare non-preemptivă a ModX-urilor cu execuţie fixă în cadrul perioadei, FENP (Subcapitolul 10.4 şi Capitolul 5). Cum parametrii temporali (reali) ai HSCD depind de cei ai aplicaţiei încărcată la un moment dat în sistem, vor trebui respectate şi verificate următoarele relaţii:

$$\begin{aligned} T_{ex}^{HSCD} &\equiv WCET_{HSCD} = f(n,\lambda) = \\ &= 56 + 71\cdot\lambda + 15(\lambda + n) + 2\cdot n + 13\cdot\lambda(\lambda+1) + WCET_{HDIS}\ [\text{CLK}] \end{aligned} \tag{11-3}$$

pentru durata de execuţie a HSCD, care este direct dependentă de numărul total de FModX-uri ale aplicaţiei (n) şi de numărul maxim de planificări dintr-un ciclu (λ);

$$T_{ex}^{HSCD} \le 2\left(T_{pr}^{M_{pr\min}} - T_{ex}^{M_{pr\min}}\right) \tag{11-4}$$

care limitează durata de execuţie a HSCD la dublul diferenţei dintre perioada şi durata de execuţie a FModX-ului cu perioada cea mai mică din set (Capitolul 5);

$$T_{pr}^{HSCD}\Big|_{\min} = f(n,\lambda) = \left\lceil \frac{\lambda - n}{2\cdot\sum_{i=1}^{n}\frac{1}{T_{pr}^{M_i}}} \right\rceil + T_{ex}^{HSCD} \tag{11-5}$$

pentru perioada minimă a HSCD, (la rândul ei o funcţie de n şi λ). Limita superioară a perioadei este dată de condiţia de planificare online periodică (vezi Capitolul 5):

$$\sum_{i=1}^{n}\left\lceil \frac{2\left(T_{pr}^{HSCD} - T_{ex}^{HSCD}\right)}{T_{pr}^{M_i}} \right\rceil \le \lambda \tag{11-6}$$

În plus, pentru setul de FModX-uri va trebui verificată (în faza de analiză, premergătoare încărcării în sistem) condiţia de necesitate a planificării FENP:

$$T_{ex}^{M_i} + T_{ex}^{M_j} \le GCD\left\{T_{pr}^{M_i}, T_{pr}^{M_j}\right\}, \quad \forall\, i,j = 1..n,\ i \ne j \tag{11-7}$$

adică, suma duratelor de execuţie a oricăror două FModX-uri să nu depăşească valoarea celui mai mare divizor comun al perioadelor acestora.

Reamintim faptul că pentru această variantă de planificator online, nucleul HARETICK utilizează faza de pre-planificare (în context SRT) pentru determinarea planificărilor primului ciclu.

11.2.1 Generarea unui semnal rectangular perfect periodic

Aplicaţia descrisă în subcapitolul anterior este utilizată într-o formă similară din punct de vedere funcţional (Figura 11-2) pentru generarea unui semnal rectangular, perfect periodic, măsurabil cu ajutorul osciloscopului la un pin de ieşire al platformei ţintă Motorola DSP56307 EVM.

Din punct de vedere al comportării temporale însă, valorile parametrilor task-urilor M_1 şi M_2 (care sunt FModX-uri de data aceasta), vor trebui specificate şi analizate conform cu parametrii semnalului rectangular dorit şi cu tipul planificării utilizate (FENP).

De asemenea, parametrii temporali ai planificatorului HSCD vor fi specificaţi în aşa fel încât să rezulte un sistem fezabil şi cât mai eficient posibil. Ca urmare, pentru HSCD vor trebui determinate o durată de execuţie suficient de mică şi o perioadă suficient de mare. Din (11-3) rezultă că T_{ex}^{HSCD} depinde de pătratul lui λ. Considerând $\lambda = 20$ (numărul maxim de planificări într-un ciclu), şi având pe $n = 2$ (numărul total de FModX-uri ale aplicaţiei), durata de execuţie a HSCD va fi:

$$WCET_{HSCD} = 7297 + WCET_{HDIS}\,[\text{CLK}] = 7297 + 486\,[\text{CLK}]$$

Exprimată în unităţi RTC (conform echivalenţei dată de (11-1)), durata de execuţie a HSCD va putea fi stabilită la valoarea:

$$T_{ex}^{HSCD} \equiv WCET_{HSCD} = 2000\,[\text{RTC}] \tag{11-8}$$

Pentru specificarea perioadei HSCD este necesară determinarea parametrilor temporali ai FModX-urilor aplicaţiei (M_1 şi M_2), astfel încât să se respecte condiţia (11-4). După implementarea codului pentru M_1 şi M_2, calculul timpului maxim (efectiv) de execuţie al ambelor FModX-uri a generat valoarea:

$$WCET_{M1}\big|_{\text{efectiv}} = WCET_{M2}\big|_{\text{efectiv}} = 35 + WCET_{HDIS} = 35 + 486\,[\text{CLK}]$$

Rezultă că durata maximă de execuţie ce poate fi specificată pentru M_1 şi M_2 nu poate fi mai mică decât 131 [RTC]. Astfel, pentru parametrii temporali ai FModX-urilor M_1 şi M_2 s-au ales valorile:

$$\begin{cases} T_{pr}^{M_1} = T_{pr}^{M_2} = 3000\,[\text{RTC}] \\ T_{ex}^{M_1} = 400\,[\text{RTC}] \\ T_{ex}^{M_2} = 200\,[\text{RTC}] \end{cases} \tag{11-9}$$

Conform (11-5), perioada minimă HSCD (durata unui ciclu de planificare) este:

$$T_{pr}^{HSCD}\Big|_{\min} = \left\lfloor \frac{20 - 2}{2\left(1/3000 + 1/3000\right)} \right\rfloor + 2000 = 15500\,[\text{RTC}]$$

Punând $T_{pr}^{HSCD} = 18000\,[\text{RTC}]$, se verifică şi condiţia de planificare online periodică (11-6), care fixează o valoare maximă pentru perioada HSCD.

Tabelul 11-4. Parametrii de implementare ai aplicaţiei şi HSCD

Denumire	Descriere	Dimensiune [24-bit words]	WCET [CLK]
HSCD_List	Listă ordonată pentru memorarea planificărilor normale (exceptând cea a HSCD) dintr-un ciclu	400	–
HSCD	Planificatorul online (algoritm FENP)	164	8000
App_Init	Task-ul (SRT) de iniţializare a aplicaţiei	17	–
M1	Setează pinul de ieşire (`s_led_on`)	32	1600
M2	Resetează pinul de ieşire (`s_led_off`)	32	800

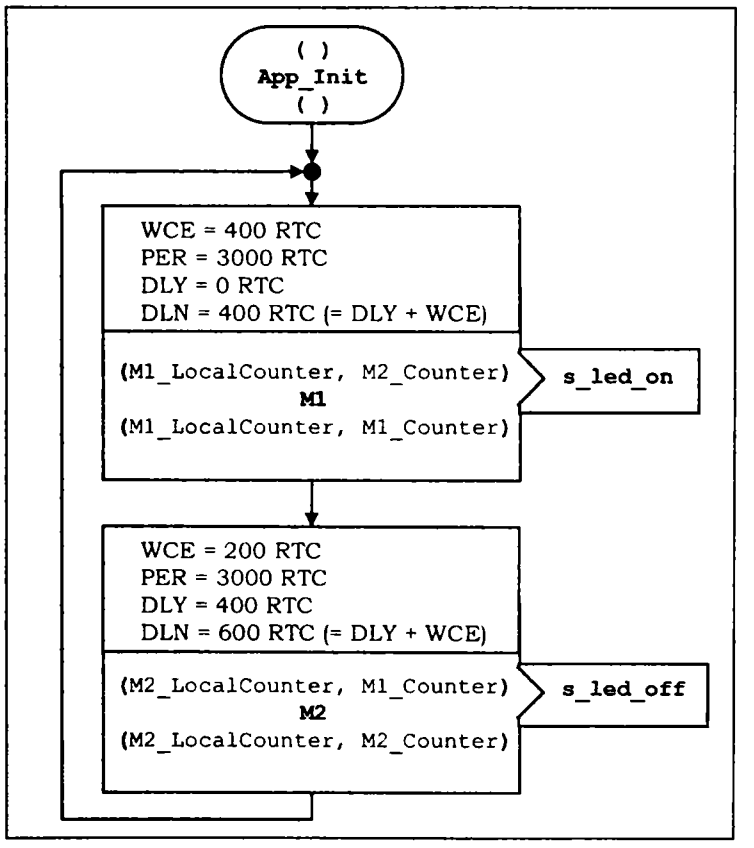

Figura 11-6. Aplicaţia de generare a unui semnal rectangular periodic

În acest punct, având toţi parametrii temporali necesari, se poate trece la etapa de analiză a planificabilităţii setului extins de FModX-uri: $\boldsymbol{M}^S \equiv \{M_1, M_2, HSCD\}$. În primul rând, condiţia de necesitate a planificabilităţii, (11-7), este validată pentru toate perechile de FModX-uri din $\boldsymbol{M}^S$. Figura 11-7 prezintă analiza planificabilităţii aplicaţiei şi diagrama de timp a semnalului periodic generat.

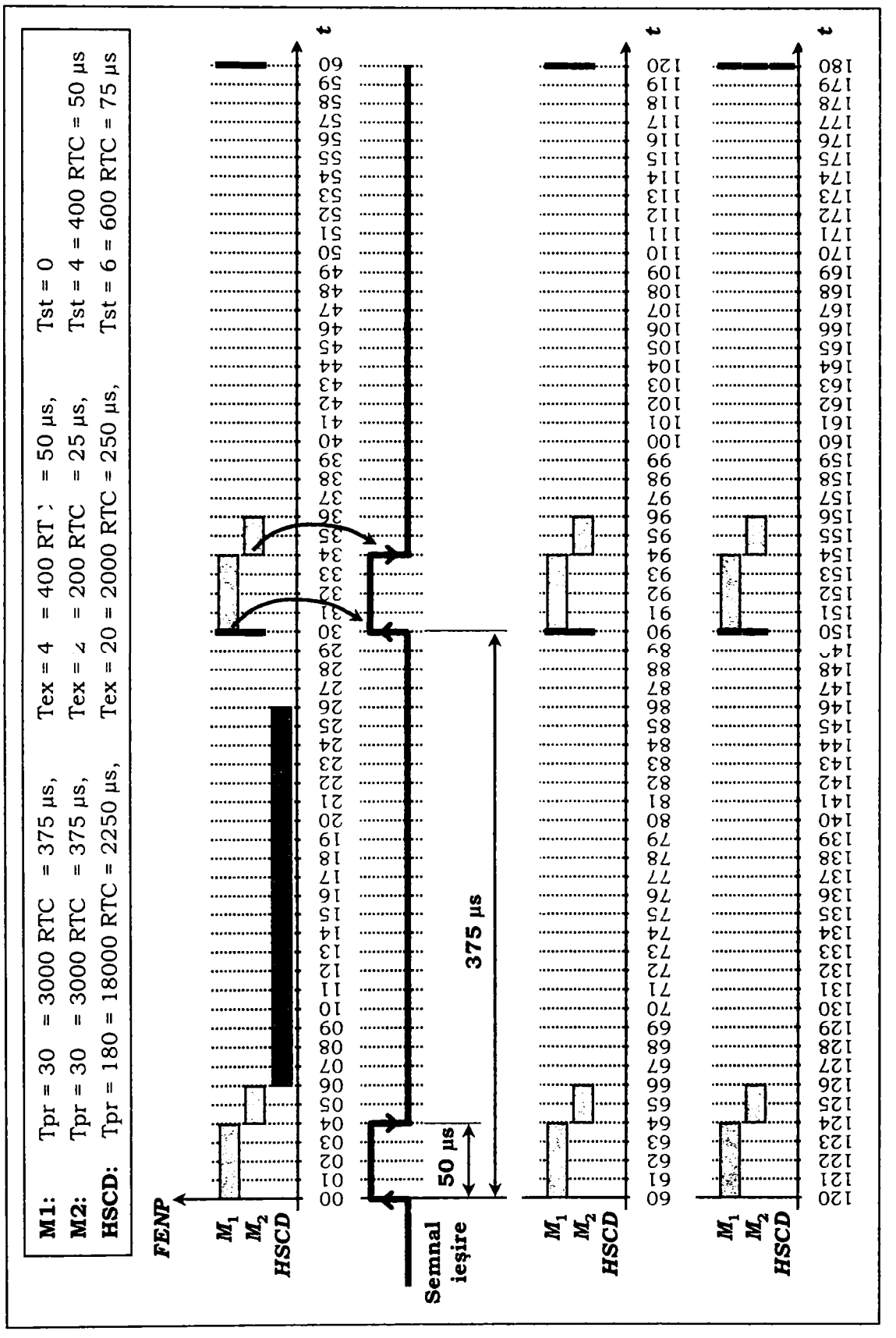

Figura 11-7. Analiza planificabilităţii aplicaţiei

Un alt rezultat al analizei planificabilităţii îl reprezintă determinarea valorilor timpilor de start ai FModX-urilor în cadrul perioadelor acestora: $T_{st}^{M_1} = 0\,[\text{RTC}]$, $T_{st}^{M_2} = 400\,[\text{RTC}]$ şi $T_{st}^{HSCD} = 600\,[\text{RTC}]$. Graful aplicaţiei, cu valorile finale ale parametrilor temporali pentru FModX-uri, este prezentat în Figura 11-6.

Utilizarea procesor a setului de FModX-uri ale aplicaţiei, $\boldsymbol{M} \equiv \{M_1, M_2\}$, are valoarea $U^{M} = 0{,}20$, iar utilizarea procesor a setului extins de FModX-uri are valoarea $U^{Ms} = 0{,}311$.

În Figura 11-7 se poate observa configuraţia planificărilor pentru primul ciclu, rezultate în urma execuţiei HSCD în regim de pre-planificare (P-HSCD): prima planificare a lui M_1 (la momentul 0), a lui M_2 (la momentul 4) şi planificarea lui HSCD (la momentul 6). Aşadar primul ciclu totalizează două planificări de FModX-uri ale aplicaţiei şi o planificare HSCD la final. Restul ciclurilor (rezultate ca urmare a execuţiilor normale HSCD, în cadrul contextului strict al nucleului) sunt compuse din câte 6 planificări ale M_1, 6 planificări ale M_2 şi o planificare HSCD la sfârşitul fiecărui ciclu. Prin urmare, există o discrepanţă între configuraţia (şi caracteristicile de timp) a ciclurilor de planificare considerate în faza de specificare a aplicaţiei, faţă de situaţia reală din timpul operării sistemului. Comparaţia dintre cele două situaţii este prezentată în Tabelul 11-5.

Tabelul 11-5. Comparaţie între parametrii temporali specificaţi şi cei efectivi, pentru FModX-urile sistem

Denumire	**WCET specificat**		**WCET efectiv**	
	[RTC]	**[µs]**	**[RTC]**	**[µs]**
P-HSCD	–	–	207	25,875
HSCD	2000	250	910	113,750
App_Init	–	–	–	–
M1	400	50	131	16,375
M2	200	25	131	16,375

Valorile efective ale WCET pentru P-HSCD şi HSCD diferă de cea specificată pentru HSCD ($n = 2$, $\lambda = 20$), prin faptul că în cazul P-HSCD avem $\lambda_{efectiv} = 2$ (două planificări de FModX-uri ale aplicaţiei) iar pentru HSCD, $\lambda_{efectiv} = 12$.

Rezultatele măsurătorilor efectuate cu ajutorul osciloscopului digital tip HAMEG "Analog and Digital Scope, HM1507-3" sunt prezentate în Figura 11-8 la Figura 11-11. Semnalul periodic generat de aplicaţia descrisă în aceste paragrafe a fost comparat cu un semnal cu aceleaşi specificaţii, obţinut cu ajutorul generatorului de funcţii programabil tip HAMEG "Programmable Function Generator, HM8130". Rezultatele obţinute atestă faptul că întregul sistem HARETICK, împreună cu aplicaţia dezvoltată şi rulată pe acesta se comportă conform cerinţelor impuse în faza de proiectare şi specificare, operarea sistemului fiind caracterizată în acelaşi timp printr-o predictibilitate maximă.

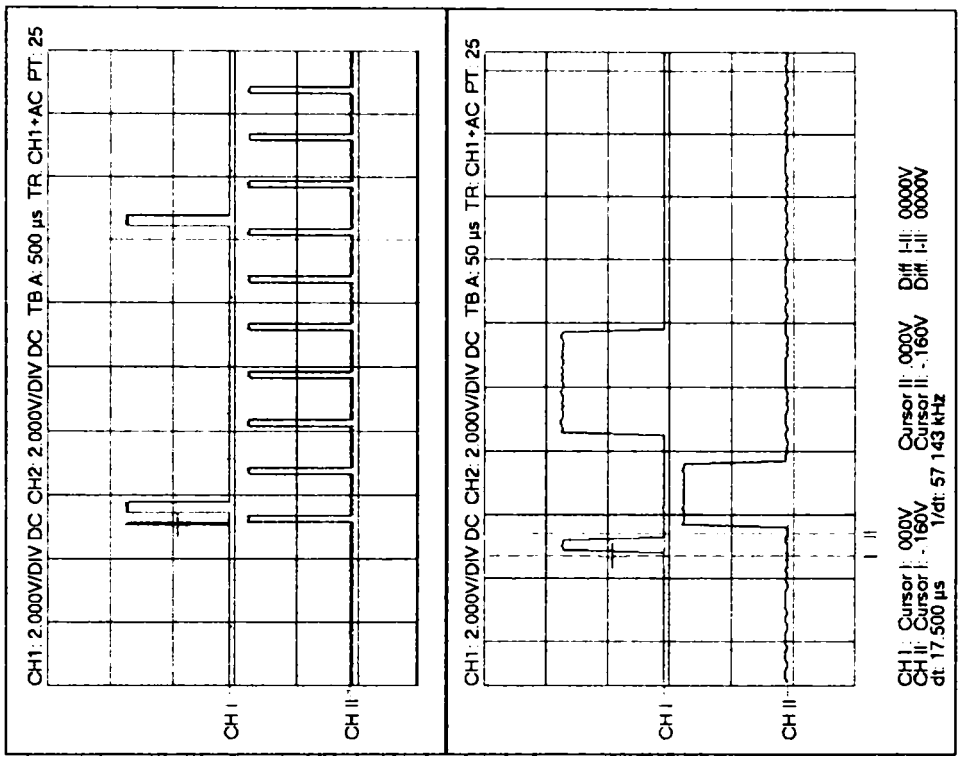

Figura 11-8. Momentul de start al operării aplicației (stânga), cu vedere de detaliu a execuției HSCD în regim de pre-planificare (dreapta). Canalul I (CH I) reprezintă execuția HSCD, iar canalul II (CH II) reprezintă semnalul periodic generat

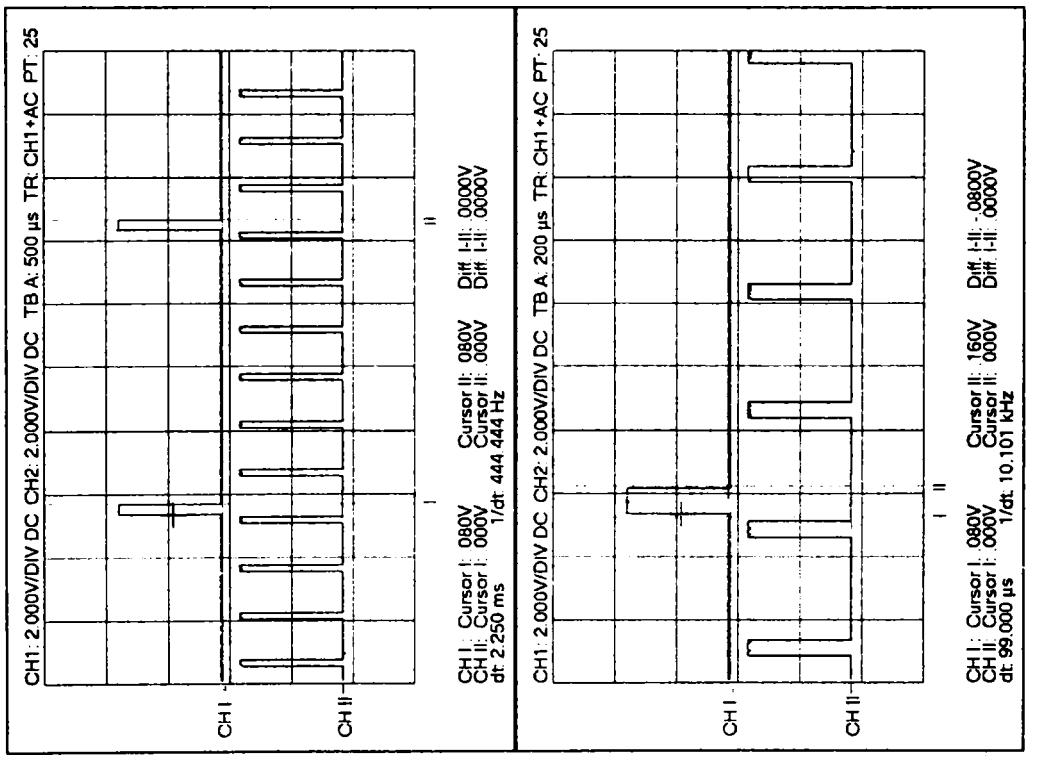

Figura 11-9. Perioada HSCD (durata unui ciclu de planificare) (stânga) şi durata de execuție a HSCD (dreapta)

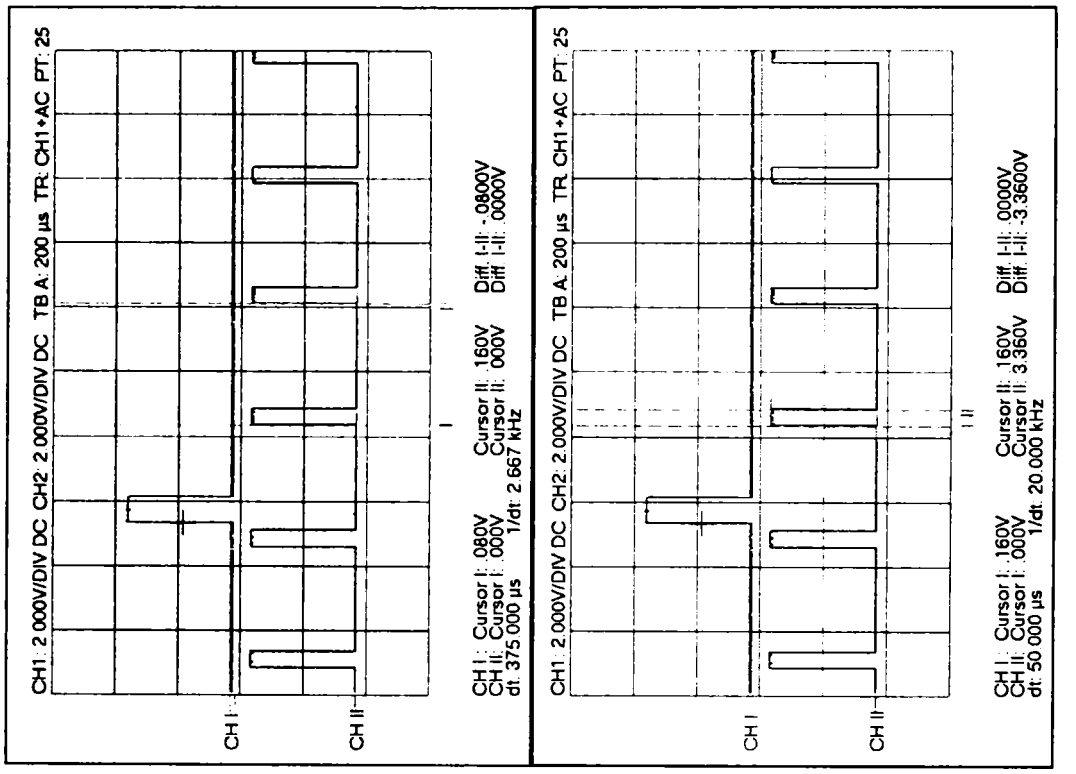

Figura 11-10. Perioada (stânga) şi lăţimea impulsului (dreapta) pentru semnalul rectangular generat

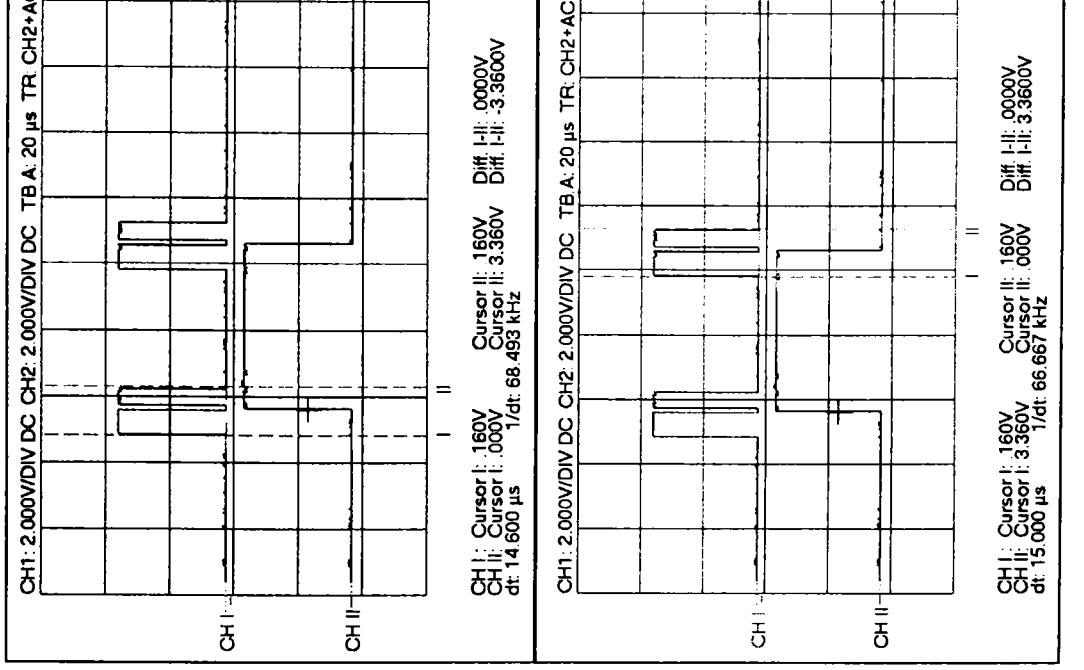

Figura 11-11. Duratele de execuţie ale FModX-urilor M_1 (stânga) şi M_2 (dreapta). Canalul I (CH I) vizualizează operarea executivului HDIS al nucleului, cu cele două componente ale sale: prefix (HDIS_PRE) şi sufix (HDIS_SUF), care încadrează execuţia FModX-urilor.

11.2.2 Evaluarea operării ModX-urilor fantomă

ModX-urile fantomă reprezintă modalitatea utilizată de nucleul HARETICK pentru a rezolva problemele legate de execuţia discontinuă a task-urilor şi de tratarea sau generarea semnalelor cu un număr de apariţii finit, dat. Un ModX oarecare ce operează în cadrul sistemului devine ModX fantomă în momentul în care contorul său de execuţii are valoarea 0 (Execution Count, `ECT` = 0). ModX-urile fantomă sunt planificate de către planificatorul online HSCD, dar nu sunt lansate în execuţie de către executivul HDIS.

Pentru testarea şi evaluarea comportării nucleului HARETICK în cazul ModX-urilor fantomă a fost dezvoltată şi implementată o variantă adaptată a aplicaţiei de generare a unui semnal periodic, discutată în paragrafele precedente. Aplicaţia, al cărui graf este prezentat în Figura 11-12, conţine două FModX-uri (M_1 şi M_2), în care M_1, pe lângă rolul de setare a semnalului de ieşire, are şi sarcina de a transforma pe M_2 în ModX fantomă şi invers, în mod consecutiv. Pentru aceasta, M_1 dispune de un parametru circular (în acelaşi timp, parametru de intrare şi de ieşire pentru acest ModX), `M1_Semafor`, căruia îi modifică alternativ valoarea, între "0" şi "1". Dacă `M1_Semafor` este găsit cu valoarea "0" la începerea unei instanţe de execuţie a lui M_1, acesta setează contorul de execuţii al lui M_2 pe 0: `M2.ECT` ← `0`, transformându-l în ModX fantomă. În celălalt caz, M_1 setează contorul lui M_2 pe "execuţie continua": `M2.ECT` ← `CONT`, transformându-l pe acesta din nou în FModX normal.

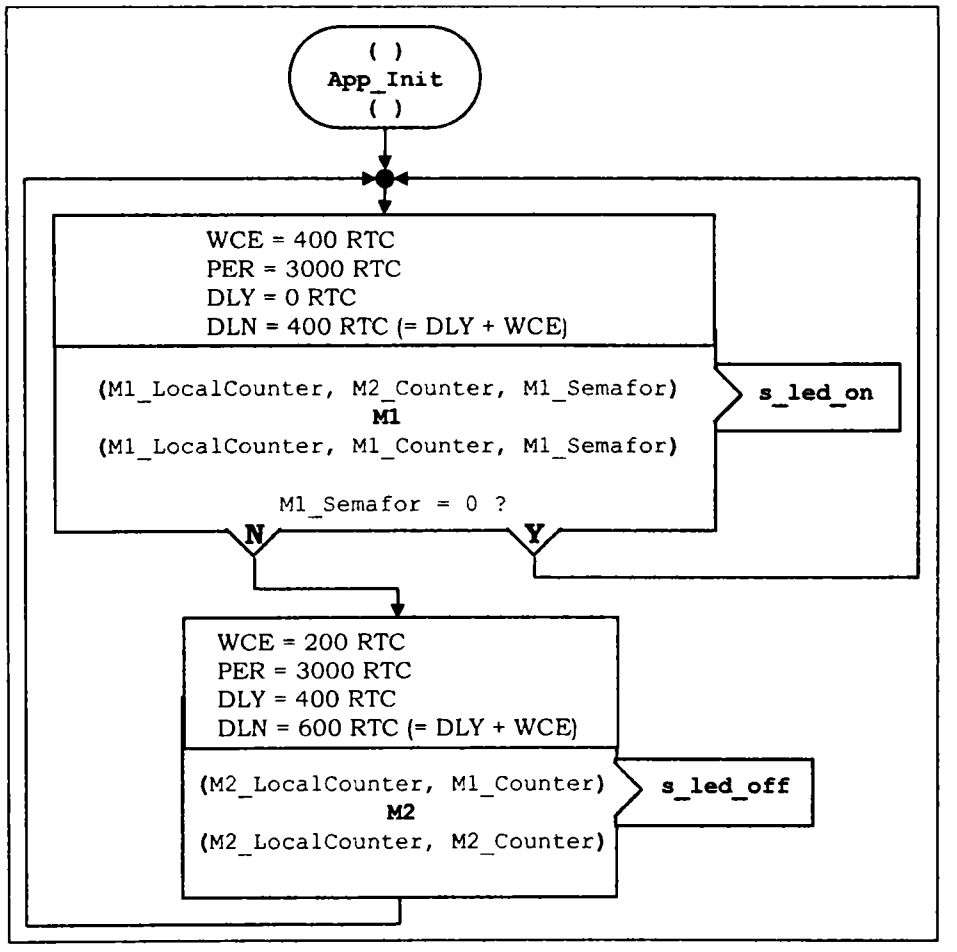

Figura 11-12. Graful aplicaţiei de evaluare a ModX-urilor fantomă

Analiza aplicaţiei este similară cu cea descrisă în Subcapitolul precedent (Figura 11-7), iar parametrii temporali ai FModX-urilor M_1, M_2 şi ai HSCD sunt, de asemenea, identici cu cei ai aplicaţiei precedente. Semnalul generat la pinul de ieşire al platformei Motorola DSP56307 EVM este tot un semnal perfect periodic, având însă perioada de 750 μs şi durata impulsului de 425 μs. Aceste caracteristici ale semnalului de ieşire se datorează faptului că pinul corespunzător este resetat doar din două în două planificări ale lui M_2 (vezi Figura 11-7).

Câteva din măsurătorile efectuate cu ajutorul osciloscopului sunt prezentate în Figura 11-13 şi Figura 11-14. Rezultatele măsurate confirmă şi pentru această aplicaţie comportarea corectă a întregului sistem HARETICK, conform cu parametrii specificaţi la proiectare şi cu valorile obţinute în faza de analiză offline.

În cele două figuri se poate observa şi execuţia planificatorului online HSCD, prin apariţia semnalului generat de executiv (canalul I – CH I, pentru partea stângă a figurilor) atunci când se termină un ciclu de planificare. Perioada HSCD este, ca şi în cazul aplicaţiei precedente, de 2250 μs.

Figura 11-14 ilustrează în mod sugestiv comportarea sistemului şi a executivului HDIS în momentul în care un ModX fantomă este planificat pentru execuţie: componenta prefix (HDIS_PRE) apelează în mod direct componenta sufix (HDIS_SUF) atunci când determină că valoarea contorului de execuţii al ModX-ului ce urmează a fi lansat în execuţie este nulă.

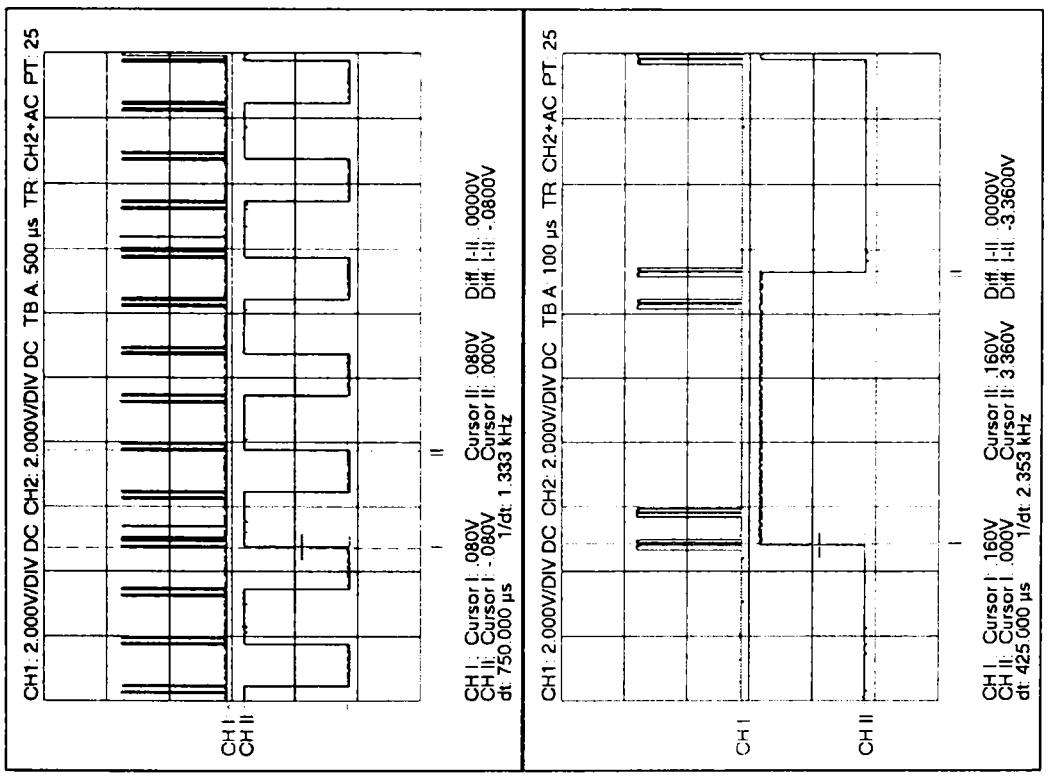

Figura 11-13. Perioada (stânga) şi durata impulsului (dreapta) pentru semnalul periodic generat. Canalul II (CH II) vizualizează semnalul de ieşire, iar canalul I (CH I) vizualizează operarea executivului HDIS al nucleului, cu cele două componente ale sale: prefix (HDIS_PRE) şi sufix (HDIS_SUF)

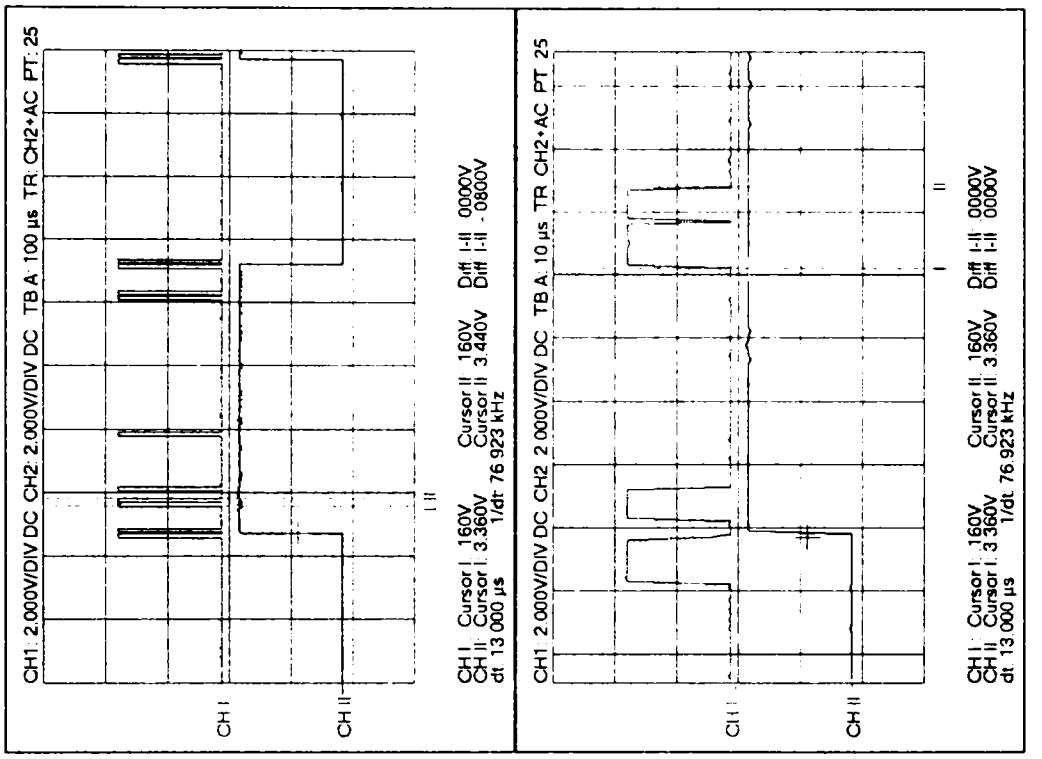

Figura 11-14. Operarea sistemului în cazul ModX-ului fantomă M_2 (stânga), cu vedere de detaliu (dreapta)

11.3 Concluzii

O variantă preliminară, dar complet funcțională a nucleului HARETICK a fost implementată şi testată cu succes pe sistemul Motorola DSP56307 EVM. Evaluarea operării nucleului a fost realizată cu ajutorul unor aplicații special proiectate şi dezvoltate. A fost vizată întâi operarea corectă a executivului TR al nucleului, HDIS, şi apoi a întregului sistem.

Rezultatele măsurătorilor realizate cu ajutorul osciloscopului digital şi prin comparare cu semnale obținute de la generatoare digitale de funcții programabile, atestă atât corectitudinea comportării nucleului din punct de vedere computațional şi temporal, cât şi faptul că modelele de semnale şi task-uri şi algoritmii de planificare introduşi şi studiați teoretic în Secțiunea II, se verifică din punct de vedere practic.

Aplicațiile descrise în acest capitol demonstrează faptul că este posibilă obținerea de semnale de ieşire perfect periodice, atingându-se nivelul maxim de predictibilitate pentru întregul sistem – unul din obiectivele principale propuse în cadrul cercetării.

Întreg procesul de proiectare, dezvoltare, analiză, implementare şi testare a aplicațiilor este relativ complicat şi puțin flexibil. Astfel, de exemplu, modificarea perioadei (sau a duratei de impuls) pentru semnalele generate, mai ales în sensul micşorării valorilor, este foarte dificilă, în special pe parcursul operării aplicației. De asemenea, nu toate configurațiile posibile (dorite) pentru parametrii unui semnal se pot realiza pe platforma HARETICK. Motivul principal al acestui inconvenient

constă din faptul că planificatorul online HSCD este la rândul său un task executat în context non-preemptiv (ModX) şi că durata sa de execuție are o valoare relativ mare în comparație cu cea a celorlaltor ModX-uri din sistem.

Rezultatele testelor efectuate relevă şi faptul că duratele de execuție reale, măsurate cu ajutorul osciloscopului, pentru ModX-uri şi chiar pentru executivul HDIS, diferă (sunt mai mici) comparativ cu valorile calculate în faza de analiză a sistemului. Acest lucru reprezintă o confirmare a faptului că în faza de analiză offline, duratele maxime de execuție (WCET) ale ModX-urilor se bazează pe evaluări pesimiste (calea cea mai lungă cale de execuție, numărul maxim de cicluri pentru bucle, etc.). Ca o consecință, deşi sistemul HARETICK operează cu predictibilitate maximă, programatorul/inginerul de sistem nu se poate baza pe momentele de terminare a execuției ModX-urilor, ci doar pe cele de start. Astfel, de exemplu, pentru o aplicație de generare a unui semnal perfect periodic, sunt necesare două ModX-uri: unul pentru stabilirea frontului crescător al semnalului şi celălalt pentru stabilirea frontului descrescător. Această abordare contrastează cu cea relativ clasică, ce utilizează un singur task care setează semnalul la începutul secvenței proprii de cod şi apoi resetează semnalul înainte de terminarea task-ului.

SECŢIUNEA IV. CONCLUZII FINALE ŞI PERSPECTIVE

Secţiunea finală a tezei conţine concluziile desprinse pe parcursul tratării subiectelor propuse în lucrare, un studiu comparativ cu alte sisteme timp-real din domeniu, rezumatul contribuţiilor şi perspectivele de continuare a cercetării în această arie tematică.

12 Concluzii finale şi perspective

12.1 Concluzii

În lucrarea de faţă se abordează *problematica sistemelor timp-real (TR) stricte destinate aplicaţiilor critice de achiziţie şi prelucrare numerică a semnalelor (APNS) şi de control digital încorporat.* Principalele idei pe care se bazează teza sunt:

- Utilizarea întreruperilor în sistemele TR stricte reprezintă o problemă din punctul de vedere al predictibilităţii, afectând capacitatea acestora de a garanta în orice condiţii de operare respectarea termenelor impuse task-urilor TR stricte;
- Pentru a oferi predictibilitate maximă sistemelor TR stricte destinate aplicaţiilor critice de achiziţie şi prelucrare numerică a semnalelor şi de control digital încorporat, este necesară abordarea modelelor non-preemptive în ceea ce priveşte definirea semnalelor, a task-urilor şi conceperea algoritmilor de planificare;
- Dezvoltarea viabilă a sistemelor şi aplicaţiilor TR stricte trebuie să aibă la bază, pentru toate fazele sale, metode şi modele omogene, compatibile, în care timpul este o coordonată cheie;
- În procesul de dezvoltare a aplicaţiilor TR stricte, faza de analiză offline, apriori execuţiei sale pe sistemul ţintă, reprezintă o cerinţă obligatorie.

Obiectivele propuse în cadrul acestui program de cercetare sunt strâns legate de problemele curente ale domeniului STR pentru aplicaţii critice de APNS şi de control digital încorporat, şi se aliniază la ideile prezentate anterior. O discuţie detaliată a modului în care au fost tratate şi atinse aceste obiective este prezentată în continuare.

(1) *Conceperea, dezvoltarea şi demonstrarea unui set omogen de modele pentru semnale şi task-uri TR, bazate pe un sistem corespunzător de reprezentare a timpului real.*

Un model al timpului real, cu rolul de element de bază pentru toate fazele dezvoltării STR stricte, a fost propus şi analizat, plecând de la principiul de operare şi sincronizare a sistemelor digitale: tactul procesor. Principalele caracteristici ale modelului temporal propus sunt:

- domeniul temporal are o structură liniară, discretă şi limitată la stânga;
- domeniul temporal poate fi echivalat cu mulţimea numerelor naturale;
- unitatea de timp are o corespondenţă directă cu intervalul din domeniul temporal absolut stabilit de ceasul de timp-real (*RTC*) al sistemului;

- modelul temporal este prevăzut cu o metrică, rezultând posibilitatea exprimării de relaţii cantitative între instanţe sau intervale temporale;
- metrica pentru timp permite determinarea duratei intervalelor temporale;
- există o relaţie bine stabilită între domeniul temporal sistem şi cel absolut.

Comportarea temporală a semnalelor are impact direct asupra comportării în timp a STR, şi reciproc. Semnalele au fost studiate şi clasificate, şi, pe baza modelului temporal introdus, a fost definit şi demonstrat un set minimal şi complet de parametri care modelează comportarea temporală a semnalelor:

- perioada (sau durata minimă dintre apariţiile consecutive ale unui semnal, în cazul în care acesta nu e periodic);
- durata de activitate (intervalul de timp în care au loc apariţiile unui semnal);
- numărul de apariţii;
- intervalul de răspuns al sistemului la apariţia unui semnal.

Comportarea în timp a unui task TR generic a fost studiată şi modelată cu ajutorul unui set de parametri şi a relaţiilor stabilite între aceştia. În continuare, au fost evidenţiate avantajele şi problemele ridicate de tratarea semnalelor de intrare cu ajutorul task-urilor periodice (utilizând tehnica de *polling*).

Sintetizând studiul semnalelor şi al task-urilor pe baza modelului temporal considerat, a fost introdus şi discutat modelul task-ului TR strict, ModX-ul:

- Task-urile periodice, executate în context non-preemptiv oferă predictibilitate maximă sistemelor TR.
- Abordarea modulară uşurează concepţia, specificarea şi proiectarea aplicaţiilor.
- Sub aspect funcţional, este necesară impunerea de restricţii la programarea ModX-urilor: evitarea recursivităţii, limitarea în timp şi spaţiu a buclelor de program, etc. Obiectivul major îl reprezintă facilitarea analizei temporale de program şi determinarea timpului maxim de execuţie a modulului (WCET).
- Comunicaţia între modulele aplicaţiei se realizează prin intermediul parametrilor de intrare/ieşire şi, într-o manieră limitată, prin variabilele globale. Datorită execuţiei ModX-urilor în context non-preemptiv, acestea implementează practic operaţii atomice, eliminându-se astfel necesitatea unor mecanisme suplimentare de sincronizare şi control al acceselor concurente la resurse comune.
- Tratarea şi generarea semnalelor este văzută ca un element important al operării sistemelor TR. Caracteristicile temporale ale ModX-urilor sunt în relaţii directe de legătură cu parametri similari ai semnalelor.
- Setul de parametri care definesc comportarea temporală a modelului de ModX propus, conţine următoarele elemente: perioada, durata de execuţie (considerată ca fiind timpul maxim de execuţie a secvenţei de cod a ModX-ului – WCET), intervalul limită (deadline), intervalul de întârziere (delay), contorul de execuţie.

Două noţiuni relativ noi au fost introduse odată cu definirea modelului de ModX. "Contorul de execuţie" al unui ModX este un parametru ce specifică şi controlează numărul ciclurilor (perioadelor) sale de execuţie. Valoarea contorului de execuţie poate fi setată pe de o parte de către programatorul de sistem/aplicaţie (în

faza de specificare), iar pe de altă parte, de către alte ModX-uri în timpul operării aplicației. De asemenea, în cazul în care se specifică un număr finit de execuții pentru un anumit ModX, executivul sistemului TR propus va decrementa contorul ModX-ului de fiecare dată când îl lansează în execuție, până se ajunge la valoarea 0. Cea de-a doua noțiune introdusă este "ModX-ul fantomă", şi reprezintă cazul ModX-urilor a căror contor de execuții ajunge la un moment dat la valoarea 0 în timpul operării. ModX-urile fantomă sunt planificate în continuare, dar nu vor fi lansate propriu-zis în execuție.

(2) *Conceperea unei metodologii unitare de dezvoltare şi implementare a sistemelor TR pentru aplicații critice de achiziție şi prelucrare numerică de semnal. Elementul central al tuturor fazelor implicate (proiectare, specificare, verificare, analiză, implementare şi testare) este sistemul de modele realizat în cadrul primului obiectiv.*

Sistemul OPEN-HARTS (Operating Environment for Hard Real-Time Systems) este conceput ca un mediu de dezvoltare şi operare pentru sisteme şi aplicații TR stricte, bazat pe setul de modele pentru timpul real, semnale şi task-uri, dezvoltat în lucrare. OPEN-HARTS are două componente majore:

- Mediul integrat, vizual, INVERTA (Integrated Visual Environment for Real-Time Application Analysis and Development), destinat dezvoltării, specificării şi verificării formale, programării, şi analizei aplicațiilor TR stricte, în regim offline:
  - Exploatând principiul modularizării, aplicațiile vor fi proiectate într-o manieră vizuală, prin construirea grafului direcționat al aplicației, unde fiecare nod reprezintă un task TR lejer (fără restricții temporale) sau un task TR strict (cu specificații temporale ce trebuie respectate şi garantate pe durata operării, în orice condiții) – ModX-ul;
  - Specificarea aplicației constă din stabilirea parametrilor fiecărui task: parametrii de intrare/ieşire şi cei globali, setul semnalelor de intrare şi de ieşire (împreună cu parametrii temporali corespunzători) şi setul parametrilor temporali ai task-ului, în cazul ModX-urilor;
  - Programarea aplicației este realizată pentru fiecare task în parte (specificarea funcțională), utilizând un limbaj de programare (asamblare, C, etc.) modificat corespunzător (restricții de limitare a structurilor şi secvențelor, în timp şi spațiu, adnotări programator suplimentare, etc.);
  - Verificarea automată a specificațiilor aplicației, în special a specificațiilor temporale ale ModX-urilor;
  - Compilarea şi analiza temporală a task-urilor, pentru stabilirea duratelor maxime de execuție (WCET);
  - Analiza planificabilității setului de ModX-uri ale aplicației, prin aplicarea condițiilor de planificabilitate şi a algoritmilor de planificare non-preemptivă;
  - Pregătirea automată a aplicației pentru încărcarea în sistemul țintă (generarea structurilor de memorie, tabela cu identificatorii de proces, etc.), în cazul în care aplicația a fost analizată cu succes.

- HARETICK (Hard Real-Time Compact Kernel), un nucleu de operare timp-real, hibrid, multi-tasking şi single-user, pentru sisteme de control încorporat. Nucleul HARETICK permite executarea aplicaţiilor TR dezvoltate cu mediul INVERTA, oferind două contexte de execuţie: contextul non-preemptiv este destinat execuţiei cu predictibilitate maximă a task-urilor TR stricte ale aplicaţiei (ModX-urile), garantând respectarea termenelor specificate, iar contextul preemptiv este destinat execuţiei task-urilor TR lejere, fără a considera timpul ca o coordonată a operării acestora.

Încărcarea şi executarea unei noi aplicaţii, dezvoltate în mediul INVERTA, pe o platformă ţintă pe care rulează HARETICK se face prin înlocuirea celei care se execută în mod curent pe platforma ţintă, existând un interval de timp (bine stabilit şi controlat) în care are loc întreruperea efectivă a operării sistemului.

Conceptele introduse de către OPEN-HARTS reprezintă o soluţie pentru problemele de predictibilitate a analizei planificabilităţii non-preemptive în cazul abordărilor online, şi, prin definirea nucleului TR hibrid, se soluţionează (cel puţin parţial) problema eficienţei scăzute a execuţiei non-preemptive.

În prezent, mediul INVERTA se află încă în faza de dezvoltare. Un pas important îl reprezintă contribuţia adusă de ing. Mircea Pătcaş, prin realizarea utilitarului de analiză a codului pentru aplicaţii scrise în limbajul de asamblare al familiei Motorola DSP56300 [Motorola 96, Motorola 00]. Utilitarul este deja capabil să extragă structura generală a programului şi să realizeze partajarea acestuia în blocuri de bază ("basic block partitioning") [Micea 04b], urmând să se finalizeze analiza temporală şi calculul timpului maxim de execuţie (WCET). Pe de altă parte, o versiune prototip, funcţională, a nucleului HARETICK a fost implementată şi testată cu succes [Micea 04a, Micea 04c].

(3) *Studiul detaliat al tehnicilor şi metodelor ce permit operarea unui sistem TR strict cu garantarea respectării termenelor de timp impuse de aplicaţiile critice şi maximizarea predictibilităţii. Interesul a fost focalizat pe studiul algoritmilor de planificare non-preemptivă a seturilor de task-uri timp-real şi a mecanismelor ce trebuie dezvoltate pentru rezolvarea elementelor asincrone din operarea sistemelor TR stricte.*

Componenta principală a dezvoltărilor teoretice din teza de faţă o constituie studiul detaliat al algoritmilor de planificare non-preemptivă a seturilor de ModX-uri (task-uri TR stricte, periodice), prezentat în Capitolul 5. Pentru început s-au discutat avantajele şi dezavantajele utilizării algoritmilor de planificare non-preemptivă şi s-a arătat faptul că introducerea etapei de analiză offline a planificabilităţii setului de task-uri, cu confirmarea (sau nu) a fezabilităţii acestuia, evită problemele de predictibilitate legate de timpii de factură non-polinomială necesari efectuării acestei analize în timpul operării efective a sistemului (analiza online).

Au fost considerate mai multe cazuri, pentru care s-au aplicat diverse restricţii, plecând de la planificarea seturilor de ModX-uri simple, independente (modele apropiate de ideal) şi avansându-se către situaţii mai complexe şi mai realiste. S-a demonstrat faptul că pentru a verifica planificabilitatea unui set oarecare de ModX-uri, este suficientă analiza planificării pe un interval limitat de timp, egal cu cel mai mic multiplu comun al perioadelor ModX-urilor din set.

Pentru seturile de ModX-uri independente au fost introduşi (adaptaţi) şi studiaţi doi dintre cei mai eficienţi algoritmi de planificare non-preemptivă: *MLFNP* (Minimum Laxity First Non-Preemptive) şi *EDFNP* (Earliest Deadline First Non-Preemptive). În acelaşi context, a fost abordată problema evaluării fezabilităţii setului de task-uri utilizând condiţii de necesitate şi/sau suficienţă. Am demonstrat că pentru modelul de task-uri considerat (ModX-uri independente, cu perioadele aliniate la momentul iniţial $t_0 = 0$), condiţia a doua de necesitate enunţată în [Jeffay 91] nu este aplicabilă. O a treia condiţie de necesitate a planificării, dictată de modelul de task TR utilizat (ModX-ul), a fost introdusă şi demonstrată.

În continuare, au fost considerate cazurile STR stricte în care operează, pe lângă setul de ModX-uri ale aplicaţiei, şi un task special – planificatorul online, care utilizează o tabelă de dispatch de dimensiuni fixe, bine determinate. S-au analizat doi algoritmi de planificare online non-preemptivă: cu cicluri cu număr constant de planificări (*CEC-NPOS*) şi cu cicluri periodice (*PC-NPOS*).

A fost abordată problema modelării şi planificării seturilor de task-uri ce necesită o execuţie fixă în cadrul fiecărei perioade a acestora (pe baza noţiunii de FModX şi a modelului matematic al acestuia, ce utilizează pe aritmetica modulară şi pe funcţia treaptă unitate). Planificarea non-preemptivă a seturilor compuse din două şi mai mult de două FModXuri independente, a fost studiată pe baza funcţiei de mapare şi a proprietăţilor sale demonstrate.

În final s-au testat şi evaluat performanţele algoritmilor de planificare non-preemptivă: evaluarea comparativă a algoritmilor *EDFNP* şi *MLFNP*, evaluarea comparativă a algoritmilor de planificare online *CEC-NPOS* şi *PC-NPOS*, şi analiza performanţelor algoritmului *FENP*. Concluzia testelor este că *EDFNP* se comportă mult mai bine ca *MLFNP*, şi *PC-NPOS* faţă de *CEC-NPOS*.

Faţă de obiectivele propuse şi formulate în capitolul de introducere, pe parcursul cercetării au mai fost abordate şi rezolvate următoarele aspecte, ce contribuie la demonstrarea practică a validităţii modelelor şi tehnicilor propuse în lucrare:

(4) *Implementarea la nivel de prototip funcţional a nucleului HARETICK pe o platformă Motorola DSP56307 EVM, ca un studiu de caz pentru modelele, metodele şi algoritmii dezvoltaţi teoretic.*

Reprezentarea şi gestionarea timpului şi a task-urilor TR stricte în cadrul nucleului HARETICK a necesitat dezvoltarea unor structuri de date şi mecanisme specifice:

- timpul absolut sistem, ceasul timp-real şi timpul de planificare;
- tabela grafului direcţionat, aciclic al aplicaţiilor (PDAGT), tabela cu descriptorii de proces (PDT), tabela de dispatch (HDis_Tab), tabela de simboluri, zona de cod ($CODE_i$) şi a parametrilor de ieşire ($DOut_i$) pentru fiecare ModX *i*, zona comună a ModX-urilor în execuţie pentru variabilele interne (Heap), etc.

Pentru testarea şi evaluarea nucleului HARETICK, au fost implementate versiunile de bază ale principalelor module: SYSINIT (iniţializarea structurilor şi componentelor nucleului), HDIS (executivul timp-real), LOADER (task-ul de încărcare a aplicaţiilor şi de gestionare a memoriei). De asemenea, au fost realizate

mai multe versiuni ale planificatorului contextului strict de execuţie al nucleului, HSCD.

Executivul HDIS a fost conceput pentru a asigura execuţia temporizată a ModX-urilor şi rezolvarea comutării corecte a celor două contexte de execuţie ale nucleului – contextul strict şi cel lejer.

(5) *Testarea şi demonstrarea operării corecte a HARETICK, şi evaluarea performanţelor sale, cu ajutorul unor aplicaţii specifice.*

O primă aplicaţie a fost dezvoltată pentru testarea operării corecte a executivului HDIS. Algoritmul de planificare utilizat se bazează pe simpla programare de execuţii consecutive a două ModX-uri în cadrul fiecărui ciclu de planificare.

Două aplicaţii pentru generarea de semnale de ieşire perfect periodice au fost apoi implementate şi testate în conjuncţie cu o versiune complet funcţională a planificatorului HSCD (bazată pe algoritmul *FENP*).

Rezultatele obţinute cu ajutorul măsurătorilor practice cu echipamente specializate (osciloscop digital, generator de funcţii programabile, etc.) ilustrează operarea corectă a nucleului HARETICK, în conformitate cu modelele şi algoritmii teoretici dezvoltaţi pentru STR stricte.

Aplicaţiile dezvoltate şi testate pe platforma HARETICK demonstrează faptul că, prin posibilitatea obţinerii de semnale de ieşire perfect periodice, întregul sistem poate atinge un nivel maxim de predictibilitate – unul din obiectivele principale propuse în cadrul cercetării.

12.2 Studiu comparativ cu alte sisteme timp-real

Deşi este încă în stadiu de dezvoltare, asupra metodologiei OPEN-HARTS se pot efectua studii comparative la diferite niveluri cu alte sisteme timp-real existente. În cele ce urmează ne vom referi la avantajele şi dezavantajele sistemului OPEN-HARTS în comparaţie cu metodologia "Giotto" [Henzinger 01, Henzinger 02, Kirsch 03, Henzinger 03a, Henzinger 03b, Henzinger 03c] şi cu sistemul "Spring" [Stankovic 91a, Stankovic 91b, Stankovic 91c, Niehaus 93, Burleson 93, Stankovic 99], ambele descrise pe scurt în Capitolul 2.

Din punct de vedere al conceptelor implicate, fiecare din cele trei sisteme poate fi considerat la nivel de metodologie. Tabelul 12-1 sintetizează comparativ principalele concepte utilizate de Giotto, Spring şi OPEN-HARTS. Se poate observa faptul că OPEN-HARTS, spre deosebire de celelalte, integrează modele bine definite pentru toate cele trei elemente pe care le considerăm esenţiale în cadrul unui STR: timpul, semnalele şi task-urile timp-real.

Modul de tratare a evenimentelor asincrone diferă între cele trei metodologii. Giotto utilizează întreruperile şi mecanismul de drivere ce tratează la nivel primar evenimentele, în context non-preemptiv. Problemele de predictibilitate în situaţii de apariţie concurentă a mai multor evenimente rămân, în opinia noastră, nerezolvate corespunzător. Spring implementează un sub-sistem separat, relativ autonom, de tratare a evenimentelor asincrone şi a semnalelor de intrare-ieşire. Predictibilitatea sistemului este astfel asigurată, cu costul creşterii complexităţii (hardware şi

software). OPEN-HARTS rezolvă problemele de predictibilitate în cazul evenimentelor asincrone prin metode de polling, ModX-urile implicate fiind specificate şi analizate apriori. Dezavantajul constă din scăderea (uneori drastică) a eficienţei sistemului.

Tabelul 12-1. Comparaţie la nivel de concepte între cele trei sisteme

Concepte implementate	**Giotto** (U. of California, Berkeley)	**Spring** (U. of Massachutsetts)	**OPEN-HARTS** (DSPLabs, U. Politehnica, Timisoara)
Platforma ţintă	embedded, mono- şi multiprocesor	sisteme distribuite	sisteme APNS şi embedded, mono-procesor
Aplicaţiile ţintă	capabilităţi timp-real stricte	TR distribuite, complexe	embedded, APNS, critice
Modelul temporal	implicit, ?	?	bine definit, integrat în toată metodologia
Modelul semnalelor	porturi (senzor, actuator, task) şi drivere	semnale cu tratare lentă şi rapidă, ?	bine definit, prin parametri temporali şi relaţiile între ei
Modelul task-urilor timp-real	modulare, periodice	periodice şi aperiodice	modulare, periodice, cu set complet de parametri temporali, execuţie NP
Tratarea evenimentelor asincrone	întreruperi, drivere NP	întreruperi, sistem I/O separat	polling, Teorema tratării semnalelor de intrare cu ModX-uri
Specificarea aplicaţiilor	limbajul Giotto, ?, utilizarea "modurilor de execuţie"	SDL (?)	mediul vizual INVERTA, parametri temporali şi specificare funcţională
Programarea aplicaţiilor	limbajul Giotto, C	Spring-C	ASM, C cu restricţii şi adnotări
Analiza şi planificarea	offline, la compilare	online, "reflectivitate"	offline în INVERTA, compilare, extracţie WCET, analiză fezabilitate
Tehnicile de planificare	selecţie programator	algoritm combinat, "SSCoP"	non-preemptive pentru HRT şi clasice pentru SRT

Planificarea task-urilor în metodologia Giotto este rezolvată de către compilator, ajutat, dacă e cazul, de programator prin adnotări corespunzătoare. Giotto oferă o flexibilitate mare în acest sens, prin faptul că nu se restricţionează utilizarea unui anumit algoritm de planificare, atâta timp cât la compilare se poate genera o planificare fezabilă a aplicaţiei. Task-urile în sistemul Spring sunt planificate în funcţie de o serie de informaţii suplimentare, furnizate de programatorul aplicaţiei sau extrase la compilare: diferiţi parametri temporali (incluzând obligatoriu "termenul limită" al fiecărui task), importanţa (prioritatea), necesar de resurse, cerinţe de fiabilitate, etc. Algoritmul de planificare Spring rezolvă şi garantează

execuţia task-urilor TR critice în mod static, apriori operării în sistem, iar celelalte task-uri vor fi planificate dinamic. Metodologia OPEN-HARTS obligă programatorul să împartă task-urile aplicaţiei în task-uri stricte (ModX-uri) şi task-uri lejere încă de la faza de specificare şi programare a aplicaţiei. ModX-urile vor fi analizate din punct de vedere al planificării non-preemptive înaintea încărcării aplicaţiei compilate in sistemul ţintă. Task-urile lejere vor fi planificate în manieră clasică, dinamică, (planificare preemptivă, bazată pe priorităţi, în "time-sharing") în timpul operării aplicaţiei.

O serie de concepte utilizate în sistemele studiate ne par extrem de interesante, urmând ca pe viitor să evaluăm în ce măsură pot fi adaptate şi aplicate în OPEN-HARTS:

- "Modurile de execuţie" din Giotto: descriu câte o anumită configuraţie bine definită de execuţie a unui set de task-uri. Cu acest concept, planificarea întregului set de task-uri poate fi uşurată prin luarea în considerare doar a task-urilor corespunzătoare modului curent la un moment dat.
- "Reflectivitatea sistemului" Spring: reprezintă caracteristica sistemului (în principal a planificatorului Spring) de a lua în considerare cât mai multă informaţie utilă despre fiecare task pentru obţinerea unei planificări eficiente şi flexibile.
- "SSCoP" – Spring Scheduling Co-Processor: introducerea unui co-procesor dedicat planificării (şi eventual, celorlalte rutine şi primitive ale nucleului de operare), eliberează procesorul de aceste sarcini şi task-uri complet transparente aplicaţiei, crescând astfel viteza şi performanţele întregului sistem.

Din punct de vedere al performanţelor, o comparaţie utilă şi corectă între cele trei sisteme este dificil de realizat, din considerente ca:

- disponibilitatea tuturor sistemelor, în versiune complet operaţională şi cu toate uneltele hardware şi software aferente;
- dezvoltarea, implementarea şi evaluarea unei aceleaşi aplicaţii pe fiecare din sistemele testate, utilizând metrici identice.

Pe de altă parte, se poate considera că, în măsura în care fiecare sistem este capabil a executa, la parametrii specificaţi, setul de aplicaţii pentru care a fost conceput, scopurile sale sunt îndeplinite. Tabelul 12-2 prezintă câteva elemente comparative desprinse din studierea performanţelor sistemelor OPEN-HARTS şi Giotto, rezultate în urma studierii documentaţiei disponibile.

Tabelul 12-2. Elemente comparative legate de performanţa sistemelor Giotto şi OPEN-HARTS

Sistemul Giotto: E- şi S- Machines	**Sistemul OPEN-HARTS: Nucleul HARETICK**
– Dimensiunea şi durata de execuţie a E-codului generat se modifică în funcţie de aplicaţie (nr. de task-uri, nr. de semnale tratate, etc.).	+ Dimensiunea şi durata de execuţie a executivului HDIS au valori fixe, cunoscute apriori.

+ Se pot specifica/genera diferite tehnici de planificare prin intermediul S-codului. – Existența driverelor, ce se execută non-preemptiv într-un context general preemptiv. Rezultă potențiale probleme de predictibilitate.	– Tehnicile non-preemptive pe care se bazează întreaga concepție HARETICK au o eficiență şi flexibilitate relativ scăzute – prețul plătit în schimbul asigurării unei predictibilități maxime. + Creşterea eficienței sistemului este soluționată prin introducerea contextului lejer de execuție. – Planificatorul online HSCD este văzut de sistem ca un ModX, utilizarea procesor implicată adăugându-se la cea a aplicației. + Sistemul este capabil a genera semnale de ieşire perfect periodice.

12.3 Rezumat al contribuțiilor

Lucrarea aduce o serie de contribuții în domeniul sistemelor TR stricte pentru aplicații critice. Un scurt rezumat al acestora este prezentat în continuare.

Capitolul 4:

- definirea unui model temporal ce permite specificarea comportamentului în timp al semnalelor şi task-urilor unui STR strict (Capitolul 4);
- studiul semnalelor şi modelarea comportamentului lor temporal prin introducerea unui set minimal şi complet de parametri;
- studiul general al task-urilor TR şi a interacțiunii acestora cu semnalele;
- abordarea comportamentului în timp al task-urilor TR periodice ce tratează semnale de intrare şi demonstrarea relațiilor temporale ce derivă din această interacțiune (timpul de răspuns, etc.);
- evidențierea şi exemplificarea faptului că metodele de tip *polling* de tratare a semnalelor de intrare garantează respectarea termenelor stricte, însă în detrimentul eficienței;
- definirea modelului de task TR strict – ModX-ul, şi discutarea proprietăților şi parametrilor acestuia relativ la aspectele considerate de importanță majoră în dezvoltarea şi analiza sistemelor şi aplicațiilor TR stricte;
- introducerea a două noțiuni noi, relativ la modelul ModX-urilor: contorul de execuție şi ModX-urile fantomă.

Capitolul 5:

- evidenţierea şi exemplificarea avantajelor şi dezavantajelor planificării şi execuţiei non-preemptive a task-urilor TR;
- evidenţierea necesităţii introducerii etapei de analiză offline a planificabilităţii setului de task-uri, cu confirmarea (sau nu) a fezabilităţii acestuia, pentru evitarea problemelor de predictibilitate legate de timpii de factură nepolinomială necesari efectuării acestei analize în timpul operării efective a sistemului (analiza online);
- studiul şi exemplificarea algoritmilor de planificare non-preemptivă *MLFNP* şi *EDFNP* pentru seturi de ModX-uri simple, independente, cu demonstrarea optimalităţii lui *EDFNP*;
- abordarea problemelor de evaluare a fezabilităţii setului de task-uri utilizând condiţii de necesitate şi/sau suficienţă;
- demonstrarea faptului că pentru modelul de task-uri considerat (ModX-uri independente, cu perioadele aliniate la momentul iniţial $t_0 = 0$), condiţia a doua de necesitate enunţată în [Jeffay 91] nu este aplicabilă;
- introducerea şi demonstrarea unei condiţii suplimentare de necesitate a planificării non-preemptive, impusă de modelul setului de task-uri bazat pe ModX-uri (condiţia a treia de necesitate);
- discutarea cazurilor, mai apropiate de realitate, ale sistemelor TR stricte ce execută, pe lângă setul de ModX-uri ale aplicaţiei, şi un task special – planificatorul online, ce utilizează o tabelă de dispatch de dimensiuni fixe, bine determinate, pentru planificarea ModX-urilor;
- ţinând cont şi de caracteristicile de comportare temporală a planificatorului online, au fost concepute şi demonstrate condiţiile de necesitate ale planificabilităţii sistemului, pentru cei doi algoritmi propuşi: planificarea online în cicluri cu număr constant de execuţii (*CEC-NPOS*) şi planificarea online în cicluri periodice (*PC-NPOS*);
- prezentarea şi verificarea prin simulări şi exemple a algoritmilor *CEC-NPOS* şi *PC-NPOS*;
- abordarea problemei modelării şi planificării seturilor de task-uri ce necesită o execuţie fixă în cadrul fiecărei perioade a acestora;
- introducerea noţiunii de ModX cu execuţie fixă (FModX) şi demonstrarea unui model matematic al acestuia, bazat pe aritmetica modulară şi pe funcţia treaptă unitate;
- studierea planificării non-preemptive a seturilor compuse din două şi mai mult de două FModXuri independente, pe baza funcţiei de mapare şi a proprietăţilor demonstrate ale acesteia;
- introducerea şi demonstrarea condiţiei de necesitate a planificabilităţii non-preemptive a seturilor de FModX-uri, ajungându-se la o formulă simplă, ce specifică faptul că dacă setul este fezabil, atunci suma duratelor de execuţie a oricăror două FModX-uri nu depăşeşte valoarea celui mai mare divizor comun al perioadelor lor;
- algoritmul de planificare a FModX-urilor, *FENP*, ce utilizează funcţia de mapare a fost conceput şi exemplificat.

Capitolul 6:

- studierea şi evaluarea performanţelor algoritmilor de planificare non-preemptivă introduşi şi discutaţi în lucrare: evaluarea comparativă a algoritmilor EDFNP şi MLFNP, evaluarea comparativă a algoritmilor de planificare online CEC-NPOS şi PC-NPOS, şi analiza performanţelor algoritmului FENP. Concluzia testelor este că *EDFNP* se comportă mult mai bine ca *MLFNP*, şi *PC-NPOS* faţă de *CEC-NPOS*.

Capitolul 7:

- conceperea şi prezentarea modelului unui sistem complet care unifică toate conceptele introduse şi studiate în lucrare – OPEN-HARTS, cu două componente: INVERTA (un mediu integrat, vizual, destinat dezvoltării, specificării şi verificării formale, programării şi analizei aplicaţiilor TR stricte) şi HARETICK (un nucleu de operare TR hibrid, multi-tasking şi single-user, pentru sisteme de control încorporat, conceput pentru a oferi predictibilitate maximă de execuţie aplicaţiilor TR critice);
- evidenţierea faptului că prin conceptele introduse de către OPEN-HARTS se oferă o soluţie pentru problemele de predictibilitate a analizei planificabilităţii non-preemptive în cazul abordărilor online, şi, prin definirea nucelului TR hibrid, se soluţionează (cel puţin parţial) problema eficienţei scăzute a execuţiei non-preemptive.

Capitolele 8, 9, 10 şi 11:

- dezvoltarea şi implementarea unui prototip de nucleu de operare TR ce poate servi ca platformă pentru execuţia aplicaţiilor critice cu garantarea respectării termenelor impuse la specificare şi cu maximizarea predictibilităţii de operare: nucleul HARETICK;
- implementarea şi testarea executivului TR al HARETICK: HDIS, cu rol esenţial în operarea task-urilor TR stricte ale sistemului şi aplicaţiilor;
- demonstrarea, cu ajutorul măsurătorilor practice, experimentale, a faptului că operarea executivului HDIS al nucleului corespunde specificaţiilor de proiectare ale sistemului;
- dezvoltarea, implementarea şi analiza unei versiuni funcţionale de planificator online pentru contextul strict de execuţie al HARETICK: HSCD;
- testarea practică şi demonstrarea, cu ajutorul măsurătorilor experimentale, a faptului că operarea planificatorului HSCD, şi, în conjuncţie cu executivul HDIS, operarea întregului sistem HARETICK, corespunde întocmai specificaţiilor de proiectare, şi de asemenea, este în concordanţă cu modelele teoretice propuse;
- implementarea şi testarea cu succes a unor aplicaţii practice pentru nucleul HARETICK, capabile a genera semnale de ieşire perfect periodice.

12.4 Concluzii finale

Atât în natură, cât şi în practica aplicaţiilor de control digital, există fenomene şi secvenţe de acţiuni care se desfăşoară fără a admite întreruperi. Pe de altă parte, operaţiile care permit apariţia întreruperilor în timpul execuţiei, pot fi descompuse la rândul lor, în sub-secvenţe continue, neîntreruptibile.

Teza de faţă demonstrează faptul că utilizarea tehnicilor non-preemptive, în conjuncţie cu un set omogen de modele pentru timp, evenimente şi acţiuni, este perfect posibilă din punct de vedere teoretic şi practic. Rezultatele obţinute cu sisteme ce utilizează principii non-preemptive sunt similare din punct de vedere al corectitudinii operaţiilor, cu cele ale sistemelor clasice. În schimb, atunci când e vorba de garantarea respectării termenelor de execuţie şi de predictibilitatea de operare, sistemele non-preemptive sunt singurele care se comportă corect în orice condiţii.

Prin urmare, abordarea non-preemptivă permite realizarea aplicaţiilor TR cu impact critic, astfel încât să fie respectate atât specificaţiile funcţionale, cât şi cele temporale ale acestora. Acest lucru se obţine însă cu preţul scăderii drastice a flexibilităţii şi eficienţei de operare a STR. Cele două probleme lasă deschise o multitudine de posibilităţi şi perspective de cercetare-dezvoltare în domeniu.

Problema flexibilităţii a fost atinsă în materialul de faţă prin conceperea modelului OPEN-HARTS, care permite în acelaşi timp dezvoltarea/modificarea aplicaţiilor TR şi operarea (execuţia) unei versiuni validate şi fezabile a acestora.

Din punctul de vedere al eficienţei STR, o soluţie completă a fost prezentată în teză, prin descrierea implementării şi testării unui nucleu de operare timp-real hibrid, HARETICK, ce conţine două contexte complementare de execuţie: contextul strict (HRT) şi cel lejer (SRT). Contextul strict de execuţie poate fi privit ca o "întrerupere prelungită" a celui lejer, conform unei planificări riguroase, analizată apriori din punctul de vedere al fezabilităţii (analiza "offline").

Un alt mod de a aborda problema este acela că, deasupra contextului lejer (reprezentând modul de operare tradiţional, preemptiv, al sistemelor de calcul de tip "time-sharing") se poate implementa, ca o facilitate de execuţie cu garantarea respectării termenelor şi cu predictibilitate maximă, contextul strict, a cărui întrerupere (RTC) este cea mai prioritară în sistem. În plus, pe durata execuţiei contextului strict, celelalte întreruperi sunt suspendate.

12.5 Perspective de cercetare şi dezvoltare

Problemele abordate în acest material, modelele, tehnicile şi algoritmii discutaţi (în majoritate noi sau cu adaptări corespunzătoare) permit continuarea preocupărilor şi deschid noi perspective în domeniul de mare interes al sistemelor şi aplicaţiilor TR stricte de achiziţie şi prelucrare numerică de semnal şi de control digital încorporat.

La rândul nostru, dorim continuarea cercetărilor întreprinse în cadrul acestui program de doctorat, urmând două direcţii distincte: (a) tehnici şi unelte pentru dezvoltarea şi analiza aplicaţiilor TR, şi (b) continuarea implementării nucleului HARETICK în vederea obţinerii unei variante complet funcţionale, care să poată

servi ca platformă de operare pentru cât mai multe aplicaţii practice. Cele două direcţii de perspectivă avute în vedere pentru cercetarea în domeniu sunt detaliate în continuare.

(a) Tehnici şi unelte pentru dezvoltarea şi analiza aplicaţiilor TR:
- Verificarea practică a seturilor de formule stabilite între parametrii temporali ai semnalelor şi task-urilor aplicaţiilor TR, pe baza modelului temporal propus. Aceste formule constituie fundamentul fazei de verificare şi validare formală a aplicaţiilor TR. Evaluarea necesităţii de formalizare a întregului sistem de modele, utilizând algebre şi logici temporale.
- Planificarea seturilor de ModX-uri cu dependenţe de program (cu alte cuvinte, planificarea unei aplicaţii TR complete).
- Integrarea tuturor cazurilor de planificare non-preemptivă şi conceperea unui model unificat, capabil a fi utilizat în cât mai multe aplicaţii reale.
- Rafinarea specificaţiilor, implementarea şi testarea mediului integrat de dezvoltare a aplicaţiilor TR (INVERTA): definirea unui limbaj adecvat de programare destinat descrierii funcţionale a task-urilor aplicaţiei (este vizat un subset al limbajului C, care să permită însă doar construcţii limitate în ceea ce priveşte durata de execuţie şi dimensiunile structurilor de date), implementarea unui compilator dedicat, care să permită generarea de cod maşină sub formă de biblioteci, separat, pentru fiecare task al aplicaţiei.
- Continuarea dezvoltării utilitarului de analiză temporală a programelor şi aplicaţiilor TR, astfel încât să se poată extrage informaţii cu privire la durata maximă de execuţie (WCET) a task-urilor. Integrarea acestuia în mediul INVERTA.
- Integrarea algoritmului unificat de planificare non-preemptivă a task-urilor TR stricte în mediul de dezvoltare a aplicaţiilor TR, pentru a se implementa faza de analiză (offline) a planificabilităţii aplicaţiei.

(b) Continuarea implementării nucleului HARETICK:
- Conceperea şi implementarea unei structuri statice, convenabile, destinată reprezentării eficiente a grafului aplicaţiilor ce vor fi încărcate în sistem (tabela PDAGT – Program Directed Acyclic Graph Table);
- Finalizarea implementării tuturor componentelor nucleului HARETICK;
- Înscrierea codului sistemului în memoria EPROM a platformei ţintă şi configurarea sistemului pentru a încărca secvenţa din EPROM la pornire sau după o operaţie de RESET;
- Finalizarea implementării protocolului şi interfeţelor de comunicaţie cu calculatorul gazdă (DATALINK), pentru a se putea citi din exterior rapoarte periodice ale execuţiei sistemului HARETICK;
- În momentul rezolvării cu succes a problemelor majore legate de implementarea curentă a HARETICK, în aşa fel încât sistemul să fie funcţional pe platforma Motorola DSP56307, intenţionăm portarea nucleului pe alte tipuri de platforme (de exemplu, sisteme cu microcontroller, etc.);

- Introducerea şi implementarea conceptelor celor două contexte complementare de execuţie din HARETICK în cadrul unor sisteme de operare TR existente.

Creşterea eficienţei şi flexibilităţii sistemelor TR prin utilizarea unui procesor dedicat planificărilor online ("scheduling coprocessor") este o altă abordare ce ni se pare valoroasă şi interesantă pentru cercetările viitoare în domeniu.

Bibliografie

[Acho 86] A. A. V. Acho, R. Sethi, J. D. Ullman, "Compilers: Principles, Techniques, Tools", Addison Wesley, Reading, Massachusetts, 1986.

[Albanus 91] J. Albanus, "Coding Schemes Used with Data Converters", Application Note AN-175, Burr-Brown, Inc., 1991.

[Allen 83] J. F. Allen, "Maintaining Knowledge about Time Intervals", Communications of the ACM, Vol. 26, No. 11, November 1983, pp. (832-843).

[Allen 94] J. F. Allen, G. Ferguson, "Actions and Events in Interval Temporal Logic", Technical Report, TR-URCSD 521, University of Rochester, New York, July 1994.

[Allombert 03] B. Allombert, A. Pacetti, "An Introduction to gp2c", 2003 [Online]. Available: http://pari.math.u-bordeaux.fr/pub/pari/.

[Audsley 91] N. C. Audsley, A. Burns, M. F. Richardson, A. J. Wellings, "Hard Real-Time Scheduling: The Deadline-Monotonic Approach", in Proceedings of the 8th IEEE Workshop on Real-Time Operating Systems and Software, Atlanta, USA, May 1991.

[Barbacci 86] M. R. Barbacci, J. M. Wing, "Specifying Functional and Timing Behavior for Real-time Applications", Technical Report ESD-TR-86-208, CMU/SEI-86-TR-4, Software Engineering Institute, Carnegie Mellon University, 1986.

[Baya 99] A. Baya, L. E. Anton, V. Ancusa, M. V. Micea, "Real Time Data Acquisition System for a Compound Straight Pipe", in Scientific and Technical Bulletin of the "Politehnica" University of Timisoara, Mechanics, Tom (44) 58, 1999, pp. (41-50).

[Bellini 00] P. Bellini, R. Mattolini, P. Nesi, "Temporal Logics for Real-time System Specification", ACM Computing Surveys, Vol 32, No. 1, March 2000.

[Berry 00] G. Berry, "The Esterel v5 Language Primer: Version v5_91", Centre de Mathematiques Appliquees, Ecole des Mines and INRIA, July 2000.

[Bucci 95] G. Bucci, M. Campanai, P. Nesi, "Tools for Specifying Real-time Systems", Real-time Systems, 8 (2/3), March/May 1995, pp. (117-172).

[Burleson 93] W. Burleson, J. Ko, D. Niehaus, K. Ramamritham, J. A. Stankovic, G. Wallace, C. Weems, "The Spring Scheduling Co-Processor: A Scheduling Accelerator", in Proceedings of the 1993 IEEE International Conference on Computer Design: VLSI in Computers and Processors, ICCD'93, October 1993, pp. (140-144).

[Burns 90] A. Burns, A. Wellings, "Real-Time Systems and Their Programming Languages", Addison-Wesley, 1990.

[Burns 91] A. Burns, M. Nicholson, N. Zhang, "Program Execution Time Analysis (SPIRITS Project Deliverable)", Department of Computer Science, University of York, July 1991.

[Burr-Brown 95a] Burr-Brown, Inc., "Data Conversion Products", Burr-Brown IC Databook, 1995.

[Burr-Brown 95b] Burr-Brown, Inc., "ADS774: Microprocessor-Compatible Sampling CMOS Analog-to-Digital Converter", Product Data Sheet PDS-1109F, 1995.

[Carter 04] E. F. Carter Jr., "Generating Gaussian Random Numbers", Taygeta Scientific, Incorporated, June 2004 [Online]. Available: http://www.taygeta.com/random/gaussian.html.

[Chapman 94] R. Chapman, "Program Timing Analysis", Technical Report, Dependable Computing Systems Centre, University of York, 1994.

[Chen 98] D. Chen, A. Mok, S. Baruah, "On Modeling Real-time Task Systems", Lecture Notes in Computer Science, No. 1494, Springer-Verlag, October 1998, pp. (153-169).

[Cheng 88] S. Cheng, J. A. Stankovic, "Scheduling Algorithms for Hard Real-Time Systems: A Brief Survey", in Hard Real Time Systems, J. A. Stankovic and K. Ramamritham (Eds.), IEEE Computer Society Press, 1988, pp. (150-173).

[Chouinard 02] J.-Y. Chouinard, "Notes on the Modular Arithmetics and Galois Fields", Course Notes, Design of Secure Computer Systems CSI4138/CEG4394, School of Information Technology and Engineering, University of Ottawa, Canada, September 2002.

[Chung 95] T. M. Chung, H. G. Dietz, "Language Constructs and Transformation for Hard Real-Time Systems", in ACM SIGPLAN Notices, Vol. 30, No. 11, November 1995, pp. (41-49).

[Colnaric 93] M. Colnaric, W. A. Halang, "Architectural Support For Predictability in Hard Real Time Systems", in Control Engineering Practice, No. 1, (1), February 1993, pp. (51-57).

[Cretu 99] V. Cretu, M. V. Micea, I. Guzun, V. Guzun, "Aquarius-DSP, from its Design to Applications Integration", in Transactions on

Automatic Control and Computer Science, Vol. 44 (58), No. 1, 2, Periodica Politehnica, Timisoara, Romania, 1999, pp. (5-16).

[Cretu 02] V. Cretu, L. E. Anton, M. V. Micea, A. Baya, R. Resiga, D. Chiciudean, "Applications for Experimental Study of Hydraulic Phenomena in Hydrodynamic Circuits", in Transactions on Automatic Control and Computer Science, Vol. 47 (61), Periodica Politehnica, Timisoara, Romania, 2002, pp. (65-70).

[Cretu 03] V. Cretu, T. Jurca, M. V. Micea, I. Sora, "Instrumentation and Measurement in Romania: Technical Developments at 'Politehnica' University of Timisoara", in IEEE Instrumentation & Measurement Magazine, Vol. 6, No. 3, September 2003, pp. (41-47).

[Ferrante 87] J. Ferrante, K. J. Ottenstein, J. D. Warren, "The Program Dependence Graph and Its Use in Optimisation", ACM Transactions on Programming Languages and Systems, No. 9 (3), July 1987, pp. (319-349).

[Fohler 01] G. Fohler, T. Lennvall, G. Buttazzo, "Inproved Habdling of Soft Aperiodic Tasks in Offline Scheduled Real-Time Systems Using Total Bandwidth Server", in Proceedings of the 8th IEEE International Conference on Emerging Technologies & Factory Automation, Nice, France, October 2001.

[Gai 02] P. Gai, L. Abeni, G. Buttazzo, "Multiprocessor DSP Scheduling in System-on-a-chip Architectures", in Proceedings of the 14th Euromicro Conference on Real-Time Systems (ECRTS'02), Vienna, Austria, June 2002, pp. (231-240).

[George 96] L. George, N. Rivierre, M. Spuri, "Preemptive and Non-Preemptive Real-Time Uni-Processor Scheduling", Rapport de recherche, Nr. 2966, Institut National de Recherche en Informatique et en Automatique, INRIA, Rocquencourt, France, September 1996.

[Gerber 97] R. Gerber, S. Hong, "Slicing Real-time Programs for Enhanced Schedulability", ACM Transactions on Programming Languages and Systems, Vol. 19, No. 3, May 1997, pp. (525-555).

[Ghosh 94] K. Ghosh, B. Mukherjee, K. Schwan, "A Survey of Real-Time Operating Systems", Technical Report GIT-CC-93/18, College of Computing, Georgia Institute of Technology, Atlanta, Georgia, February 1994.

[Coffman 73] E. G. Coffman Jr., P. J. Denning, "Operating Systems Theory", Prentice-Hall Series in Automatic Computation, Prentice-Hall, New Jersey, USA, 1973.

[Gupta 96] R. Gupta, M. Spezialetti, "A Compact Task Graph Representation for Real-time Scheduling", Real Time Systems, 10, 1-32 (1996), Kluwer Academic Publishers, 1996.

[Halpern 91] J. Y. Halpern, Y. Soham, "A Propositional Modal Logic of Time Intervals", Journal of the ACM, Vol. 38, No. 4, October 1991, pp. (935-962).

[Henzinger 01] T. A. Henzinger, B. Horowitz, C. M. Kirsch, "Embedded Control Systems Development with Giotto", in Proceedings of the International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES), ACM Press, 2001, pp. (64-72).

[Henzinger 02] T. A. Henzinger, C. M. Kirsch, "The Embedded Machine: Predictable, Portable Real-Time Code", in Proceedings of the International Conference on Programming Language Design and Implementation (PLDI), ACM Press, 2002, pp. (315-326).

[Henzinger 03a] T. A. Henzinger, B. Horowitz, C. M. Kirsch, "Giotto: A Time-triggered Language for Embedded Programming", in Proceedings of the IEEE, No. 91, 2003, pp. (84-99).

[Henzinger 03b] T. A. Henzinger, C. M. Kirsch, M. A.A. Sanvido, W. Pree, "From Control Models to Real-Time Code Using Giotto", in IEEE Control Systems Magazine, No. 23 (1), 2003, pp. (50-64).

[Henzinger 03c] T. A. Henzinger, C. M. Kirsch, S. Matic, "Schedule Carrying Code", in Proceedings of the Third International Conference on Embedded Software (EMSOFT), Lecture Notes in Computer Science 2855, Springer-Verlag, 2003, pp. (241-256).

[Howell 95] R. R. Howell, M. K. Venkatrao, "On Non-Preemptive Scheduling of Recurring Tasks Using Inserted Idle Times", in Information and Computation Journal, Vol. 117, No. 1, February, 1995.

[Ifeachor 93] E. C. Ifeachor, B. W. Jervis, "Digital Signal Processing: A Practical Approach", Addison-Wesley, 1993.

[Jahanian 88] F. Jahanian, A. K. Mok, D. A. Stuart, "Formal Specification of Real-time Systems", UTCS Technical Report, UTCS-TR-88-25, Department of Computer Sciences, University of Texas at Austin, 1988.

[Jeffay 91] K. Jeffay, D. Stanat, C. Martel, "On Non-Preemptive Scheduling of Periodic and Sporadic Tasks", in Proceedings of the Twelfth IEEE Real-Time Systems Symposium, San Antonio, Texas, December 1991, IEEE Computer Society Press, pp. (129-139).

[Kang 98] S.-I. Kang, H.-K. Lee, "Analysis and Solution of Non-preemptive Policies for Scheduling Readers and Writers", in ACM Operating Systems Review, No. 32 (3), July, 1998, pp. (30-50).

[Kim 80] K. H. Kim, M. Naghibzadeh, "Prevention of Task Overruns in Real-Time Non-Preemptive Multiprogramming Systems", ACM, 1980, pp. (267-276).

[Kirsch 03] C. M. Kirsch, T. A. Henzinger, M. A. A. Sanvido, "A Programmable Microkernel for Real-Time Systems", Technical Report UCB/CSD-3-1250, University of California, Berkeley, USA, June 2003.

[Klingerman 86] E. Klingerman, A. D. Stoyenko, "Real-time Euclid: A Language for Reliable Real-time Systems", IEEE Transactions on Software Engineering, SE-12 (9), September 1986.

[Knuth 98] D. E. Knuth, "The Art of Computer Programming", Vol. 2, "Seminumerical Algorithms", Addison-Wesley, Reading Massachusetts, 1998.

[Lehoczky 87] J.P. Lehoczky, L. Sha, J.K. Strosnider, "Enhanced Aperiodic Responsiveness in Hard Real-Time Environments", in Proceedings of the 8th Real-Time Systems Symposium, San Jose, California, 1987, pp. (261-270).

[Lehoczky 89] J. P. Lehoczky, L. Sha, Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior", in Proceedings of the 10th Real-Time Systems Symposium, IEEE Computer Society Press, December 1989, pp. (166-171).

[Lerma 03] M. A. Lerma, "Modular Arithmetic", Department of Mathematics, Northwestern University, USA, March 2003 [Online]. Available: http://www.math.northwestern.edu/~mlerma/problem_solving/results/modular_arith.pdf.

[Letia 00] T. S. Letia, "Sisteme de timp-real" ("Real-Time Systems"), "Editura Albastra" Publishing House, Cluj-Napoca, Romania, 2000.

[Liu 73] C. L. Liu, J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", in Journal of the ACM, Vol. 20, No 1., January 1973, pp. (46-61).

[Lynn 92] P. A. Lynn, W. Fuerst, "Introductory Digital Signal Processing with Computer Applications", John Wiley & Sons, 1992.

[Marlowe 90] T. J. Marlowe, B. G. Ryder, "Properties of Data Flow Frameworks", Acta Informatica, 28, pp. (121-163), 1990.

[Mattolini 01] R. Mattolini, P. Nesi, "An Interval Logic for Real-Time System Specification", in IEEE Transactions on Software Engineering, Vol. 27, No. 3, March 2001, pp. (208-227).

[Mattolini 96] R. Mattolini, P. Nesi, "Using TILCO for Specifying Real-Time Systems", in Proceedings of the Second IEEE International Conference on Engineering of Complex Computer Systems

(Montreal, Canada), IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. (18-25).

[Melliar-Smith 87] P. M. Melliar-Smith, "Extending Interval Logic to Real Time Systems", in Proceedings of the Conference on Temporal Logic Specification (UK), Ed. B. Banieqbal, H. Barringer, A. Pnuelli, Springer-Verlag, New York, 1987, pp. (224-242).

[Mercer 92] C. W. Mercer, "An Introduction to Real-Time Operating Systems: Scheduling Theory", Draft, School of Computer Science, Carnegie Mellon University, Pittsburg, Pennsylvania, November 1992.

[Micea 00a] M. V. Micea, "Interfacing DAQ Systems to Digital Signal Processors", Transactions on Automatic Control and Computer Science, Special Issue Dedicated to the Fourth International Conference on Technical Informatics, CONTI'2000, October 12-13, Vol. 45 (59), No. 4, "Politehnica" University of Timisoara, 2000, pp. (23-28).

[Micea 00b] M. V. Micea, V. Cretu, D. Chiciudean, "Interfacing a Data Acquisition System to the DSP56303", Application Note AN2087/D Rev. 0, 12/2000, Motorola, Inc., USA, 2000.

[Micea 00c] M. V. Micea, "Sisteme de achizitie numerica a datelor: Indrumator de laborator", ("Signal Acquisition and Conditioning Systems: Laboratory Workshops"), Comanda 270/2000, Centrul de multiplicare al Universitatii POLITEHNICA Timisoara, 2000.

[Micea 00d] M. V. Micea, V. Cretu, D. Chiciudean, "Communication Protocols Implementation on DSP-based Systems", in Proceedings of the International Symposium on Systems Theory, SINTES 10, 10-th Edition, Automation, Computers, Electronics, May 25-26, University of Craiova, 2000, pp. (C 205-208).

[Micea 00e] M. V. Micea, D. Chiciudean, L. Muntean, "ECP Standard Parallel Interface for DSP56300 Devices", Application Note AN2085/D Rev. 0, 11/2000, Motorola, Inc., USA, 2000.

[Micea 00f] M. V. Micea, V. Cretu, D. Chiciudean, "Interfacing a Data Acquisition System to the DSP56303", Application Note AN2087/D Rev. 0, 12/2000, Motorola, Inc., USA, 2000.

[Micea 01a] M. V. Micea, A. Trifu, M. Trifu, "Integrating the DSP563xx in Distributed Computing Environments", Application Note AN2088/D Rev. 0, 3/2001, Motorola, Incorporated, USA, 2001.

[Micea 01b] M. V. Micea, L. Muntean, D. Brosteanu, "Simple Real-time Sonar with the DSP56824", Application Note AN2086/D Rev. 0, 6/2001, Motorola, Incorporated, USA, 2001.

[Micea 01c] M. V. Micea, M. Stratulat, D. Ardelean, D. Aioanei, "Implementing Professional Audio Effects with DSPs", in Transactions on Automatic Control and Computer Science, Vol. 46 (60), Periodica Politehnica, Timisoara, Romania, 2001, pp. (55-60).

[Micea 04a] M. V. Micea, V. Cretu, "Non-Preemptive Execution Support for Critical and Hard Real-Time Applications on Embedded Platforms", in Proceedings of the 2004 International Symposium on Signals, Systems and Electronics, ISSSE'04, Linz, Austria, August 2004, in print.

[Micea 04b] M. V. Micea, V. Cretu, L. M. Patcas, "Program Modeling and Analysis of Real-Time and Embedded Applications", in the Scientific Bulletin of "Politehnica" University of Timisoara, Transactions on Automatic Control and Computer Science, Vol. 49 (63), No. 3, Timisoara, Romania, May 2004, pp. (207-212).

[Micea 04c] M. V. Micea, "HARETICK: A Real-Time Compact Kernel for Critical Applications on Embedded Platforms", in Proceedings of the 7th International Conference on Development and Application Systems, DAS2004, Editura Universitatii Suceava, Suceava, Romania, May 2004, pp. (16-23).

[Mok 83] A. K. Mok, "Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment", PhD. Thesis, MIT Technical Report MIT/LCS/TR-297, Laboratory of Computer Science, Massachuttes Institute of Technology, Massachuttes, May 1983.

[Mok 84] A. K. Mok, "The Design of Real-Time Programming Systems Based on Process Models", in IEEE Proceedings of the 5th Real Time Systems Symposium, Austin, Texas, December 1984, pp. (5-17).

[Motorola 00] Motorola, Inc., "DSP56300: 24-Bit Digital Signal Processor: Family Manual", Revision 3, DSP56300FM/AD, Semiconductor Products Sector, DSP Division, Austin, USA, November 2000.

[Motorola 95] Motorola, Inc., "DSP56000: 24-Bit Digital Signal Processor: Family Manual", DSP56KFAMUM/AD, Semiconductor Products Sector, DSP Division, Austin, USA, 1995.

[Motorola 96] Motorola, Inc., "Motorola DSP: Assembler Reference Manual", Version 6.0, Semiconductor Products Sector, DSP Division, Austin, USA, 1996.

[Motorola 98] Motorola, Inc., "DSP56307: 24-Bit Digital Signal Processor: User's Manual", DSP56307UM/D, Revision 0, 08/10/98, Semiconductor Products Sector, DSP Division, Austin, USA, August 1998.

[Motorola 99] Motorola, Inc., "DSP56307EVM User's Manual", Revision 1.8, 3/1999, DSP56307EVMUM/D, Semiconductor Products Sector, DSP Division, Austin, USA, March 1999.

[Muchnick 97] S. S. Muchnick, "Advanced Compiler Design and Implementation", Academic Press, Morgan Kaufmann Publishers, 1997.

[NatInst 99] National Instruments Corp., "Measurement and Automation Catalogue", 1999.

[Niehaus 91] D. Niehaus, "Program Representation and Translation for Predictable Real-time Systems", Department of Computer and Information Science, University of Massachusetts, 1991.

[Niehaus 93] D. Niehaus, K. Ramamritham, J. A. Stankovic, G. Wallace, C. Weems, W. Burleson, J. Ko, "The Spring Scheduling Co-Processor: Design, Use and Performance", in Real-Time Systems Symposium, December 1993.

[Ning 02] P. Ning, "Basic Number Theory (Topic 2.3)", Course Notes, Information Systems Security CSC 495N/574, Department of Computer Science, North Carolina State University, USA, 2002.

[Oppenheim 96] A. V. Oppenheim, R. W. Schafer, "Digital Signal Processing", Prentice-Hall, 1996.

[Ostroff 92] J. S. Ostroff, "Formal Methods for the Specification and Design of Real-time Safety Critical Systems", Journal of Systems and Software, Vol. 18, No. 1, 1992, pp. (33-60).

[Panzieri 93a] F. Panzieri, L. Donatiello, L. Poretti, "Scheduling Real Time Tasks: A Performance Study", Technical Report UBLCS-93-10, May 1993, Laboratory for Computer Science, University of Bologna, Italy.

[Panzieri 93b] F. Panzieri, R. Davoli, "Real Time Systems: A Tutorial", Technical Report UBLCS-93-22, Laboratory of Computer Science, University of Bologna, Italy, October 1993.

[Parashkevov 94] A. Parashkevov, "Distributed Real-time Computer Systems", Technical Report CRTS-94-01, Department of Computer Science, University of Adelaide, Australia, 1994.

[Park 91] C. Y. Park, A. C. Shaw, "Experiments with a Program Timing Tool Based on Source-level Timing Schema", IEEE Computer, May 1991, pp. (48-57).

[Pentek 96] Pentek, Inc., "Digital Signal Processing and Data Acquisition", Product Catalog, 1996.

[Podgurski 89] A. Podgurski, L. A. Clarke, "The Implications of Program Dependences for Software Testing, Debugging and Maintenance", ACM Software Engineering Notes, No. 14 (8), 1989, pp. (168-178).

[Proakis 96] J. G. Proakis, D. G. Manolakis, "Digital Signal Processing: Principles, Algorithms and Applications", 3rd Edition, Prentice-Hall, 1996.

[Puschner 89] P. P. Puschner, C. Koza, "Calculating the Maximum Execution Time of Real-time Programs", Journal of Real-time Systems, 1 (2), September 1989, pp. (159-176).

[Puschner 91] P. P. Puschner, A. V. Schedl, "Computing Maximum Task Execution Times – A Graph-based Approach", Kluwer Academic Publishers, 1991.

[Puschner 00] P. P. Puschner, A. Burns, "A Review of Worst-Case Execution Time Analysis", Guest Editorial, in Real-Time Systems, No. 18 (2/3), May 2000, pp. (115-127).

[QNX 99a] QNX Software Systems, Ltd., "QNX Neutrino v2 Realtime Operating System: Building Embedded Systems", 3rd Edition, Ontario, Canada, May 1999.

[QNX 99b] QNX Software Systems, Ltd., "QNX Neutrino v2 Realtime Operating System: Development Tools", 2nd Edition, Ontario, Canada, June 1999.

[Radulescu 02] A. Radulescu, A. J. C. van Gemund, "Low-Cost Task Scheduling for Distributed-Memory Machines", in IEEE Transactions on Parallel and Distributed Systems, Vol 13, No. 6, June 2002, pp. (648-658).

[Ramamritham 94] K. Ramamritham, J. A. Stankovic, "Scheduling Algortihms and Operating Systems Support for Real-Time Systems", in Proceedings of the IEEE, Vol. 82, No. 1, January 1994, pp. (55-67).

[Ranvidran 02] B. Ranvidran, "Engineering Dynamic Real-Time Distributed Systems: Architecture, System Description Language and Middleware", in IEEE Transactions on Software Engineering, Vol. 28, No. 1, January 2002, pp. (30-57).

[Robu 02] N. Robu, "Concurrent Programming. Real-Time Oriented Support Mechanisms", ("Programare concurenta. Mecanisme suport orientate timp real"), Editura "Politehnica" Timisoara, Romania, 2002.

[Shamus 03] Shamus Software Ltd., "MIRACL Users Manual", Version 4.8, Dublin, Ireland, August 2003.

[Shaw 89] A. C. Shaw, "Reasoning about Time in Higher Level Language Software", IEEE Transactions on Software Engineering, 15 (7), July 1989, pp. (875-889).

[Shoup 02] V. Shoup, "A Tour of NTL", 2002 [Online]. Available: http://www.shoup.net.

[Sprunt 89a] B. Sprunt, J.P. Lehoczky, L. Sha, "Scheduling Sporadic and Aperiodic Events in a Hard Real-Time System", Technical Report CMU/SEI-890TR-11, April 1989.

[Sprunt 89b] B. Sprunt, L. Sha, J.P. Lehoczky, "Aperiodic Task Scheduling for Hard-Real-Time Systems", in Real-Time Systems, Vol. 1, No. 1, June 1989, pp. (27-60).

[Stallings 98] W. Stallings, "Operating Principles: Internals and Design Principles", 3rd Edition, Prentice-Hall, New Jersey, USA, 1998.

[Stankovic 88] J. A. Stankovic, "Misconceptions about Real-Time Computing", IEEE Computer, Vol. 21, No. 10, October 1988, pp. (10-19).

[Stankovic 90] J. A. Stankovic, "The Spring Architecture", in Proceedings of the IEEE Workshop on Real Time, Euromicro'90, June 1990, pp. (104-113).

[Stankovic 91a] J. A. Stankovic, "On the Reflective Nature of the Spring Kernel", invited paper, in Proceedings of Process Control Systems '91, February 1991.

[Stankovic 91b] J. A. Stankovic, K. Ramamritham, "The Spring Kernel: A New Paradigm for Hard Real-Time Operating Systems", in IEEE Software, No. 8 (3), May 1991, pp. (62-72).

[Stankovic 91c] J. A. Stankovic, D. Niehaus, K. Ramamritham, "SpringNet: A Scalable Architecture for High Performance, Predictable, and Distributed Real-Time Computing", Univ. of Massachusetts, TR 91-74, October 1991.

[Stankovic 92a] J. A. Stankovic, "Distributed Real-Time Computing: The Next Generation", Invited keynote paper, Special issue of "Journal of the Society of Instrumentation and Control Engineers of Japan", Vol. 31, No. 7, pp. (726-736), 1992.

[Stankovic 92b] J. A. Stankovic, "Real-Time Computing", Invited paper, BYTE, pp. (155-160), August 1992.

[Stankovic 99] J. A. Stankovic, K. Ramamritham, D. Niehaus, M. Humphrey, G. Wallace, "The Spring System: Integrated Support for Complex Real-Time Systems", in the special issue of the Real-Time Systems Journal, Vol. 16, No. 2/3, 1999.

[Stewart 01] D. B. Stewart, "Twenty-five Most Common Mistakes with Real-time Software Development", in 2001 Embedded Systems Conference, Class 270, San Francisco, April 2001.

[Stoyenko 91] A. D. Stoyenko, C. Hamacher, R. C. Holt, "Analyzing Hard Real-time Programs for Guaranteed Schedulability", IEEE Transactions on Software Engineering, 17 (8), August 1991, pp. (737-750).

[Stoyenko 95] A. D. Stoyenko, T. J. Marlowe, M. F. Younis, "A Language for Complex Real-time Systems", in The Computer Journal, Vol. 38, No. 4, 1995, pp. (319-338).

[Tanenbaum 97] A. S. Tanenbaum, A. S. Woodhull, "Operating Systems: Design and Implementation", 2nd Edition, Prentice-Hall, New Jersey, USA, 1997.

[TexasInst 92] Texas Instruments, Inc., "TMS320C31 Embedded Control Technical Brief", SPRU083, August 1992.

[TexasInst 97] Texas Instruments, Inc., "TMS320 DSP Development Support Reference Guide", SPRU011E, January 1997.

[TexasInst 00a] Texas Instruments, Inc., "DAC707/708/709: Microprocessor-Compatible 16-Bit Digital-to-Analog Converters", Burr-Brown Product Data Sheet PDS-557H (SBAS145), 2000.

[TexasInst 00b] Texas Instruments, Inc., "DAC80/DAC80P: Monolithic 12-bit Digital-to-Analog Converters", Burr-Brown Product Data Sheet PDS-643F (SBAS148), 2000.

[TexasInst 00c] Texas Instruments, Inc., "MPC506A/507A: Single-Ended 16-Channel / Differential 8-Channel CMOS Analog Multiplexers", Burr-Brown Product Data Sheet PDS-774E (SBFS018), 2000.

[Toma 96] L. Toma, "Sisteme de achiziţie şi prelucrare numerică a semnalelor" ("Digital Signal Acquisition and Processing Systems"), Editura de Vest, Timisoara, 1996.

[Ulrich 98] R. Ulrich, "Gaussian Distribution: FAQ", 1998 [Online]. Available: http://www.pitt.edu/~wpilib/stats99.html.

[Young 82] S. Young, "Real Time Languages: Design and Development", Ellis Horwood Publishers, 1982.

Lista lucrărilor publicate

[1] M. V. Micea, V. Cretu, "Non-Preemptive Execution Support for Critical and Hard Real-Time Applications on Embedded Platforms", in Proceedings of the 2004 International Symposium on Signals, Systems and Electronics, ISSSE'04, Linz, Austria, August 2004, [CD-ROM version].

[2] M. V. Micea, V. Cretu, L. M. Patcas, "Program Modeling and Analysis of Real-Time and Embedded Applications", in the Scientific Bulletin of "Politehnica" University of Timisoara, Transactions on Automatic Control and Computer Science, Vol. 49 (63), No. 3, Timisoara, Romania, May 2004, pp. (207-212).

[3] M. V. Micea, "HARETICK: A Real-Time Compact Kernel for Critical Applications on Embedded Platforms", in Proceedings of the 7th International Conference on Development and Application Systems, DAS2004, Editura Universitatii Suceava, Suceava, Romania, May 2004, pp. (16-23).

[4] V. Cretu, T. Jurca, M. V. Micea, I. Sora, "Instrumentation and Measurement in Romania: Technical Developments at 'Politehnica' University of Timisoara", in IEEE Instrumentation & Measurement Magazine, Vol. 6, No. 3, September 2003, pp. (41-47).

[5] M. V. Micea, "Introducere in Prelucrarea Numerica a Semnalelor: Suport de curs", ("Introduction to Digital Signal Processing: Course Notebook"), Comanda 120/2003, Centrul de multiplicare al Universitatii POLITEHNICA Timisoara, 2003.

[6] V. Cretu, L. E. Anton, M. V. Micea, A. Baya, R. Resiga, D. Chiciudean, "Applications for Experimental Study of Hydraulic Phenomena in Hydrodynamic Circuits", in Transactions on Automatic Control and Computer Science, Vol. 47 (61), Periodica Politehnica, Timisoara, Romania, 2002, pp. (65-70).

[7] M. V. Micea, L. Muntean, D. Brosteanu, "Embedded Techniques for Autonomous Robot Orientation", in Proceedings of the 6-th International Conference on Development and Application Systems, DAS2002, Suceava, Romania, May 2002, pp. (22-27).

[8] M. V. Micea, M. Stratulat, D. Ardelean, D. Aioanei, "Implementing Professional Audio Effects with DSPs", in Transactions on Automatic Control and Computer Science, Vol. 46 (60), Periodica Politehnica, Timisoara, Romania, 2001, pp. (55-60).

[9] M. V. Micea, L. Muntean, D. Brosteanu, "Simple Real-time Sonar with the DSP56824", Application Note AN2086/D Rev. 0, 6/2001, Motorola, Incorporated, USA, 2001.

[10] M. V. Micea, A. Trifu, M. Trifu, "Integrating the DSP563xx in Distributed Computing Environments", Application Note AN2088/D Rev. 0, 3/2001, Motorola, Incorporated, USA, 2001.

[11] M. V. Micea, V. Cretu, D. Chiciudean, "Interfacing a Data Acquisition System to the DSP56303", Application Note AN2087/D Rev. 0, 12/2000, Motorola, Incorporated, USA, 2000.

[12] M. V. Micea, "Interfacing DAQ Systems to Digital Signal Processors", in Transactions on Automatic Control and Computer Science, Special Issue Dedicated to the Fourth International Conference on Technical Informatics, CONTI'2000, Vol. 45 (59), No. 4, Timisoara, Romania, October 2000, pp. (23-28).

[13] M. V. Micea, L. Muntean, D. Brosteanu, D. Chiciudean, "Simple Real-time SONAR Implemented with DSP-based Boards", in Proceedings of the 5-th International Conference on Development and Application Systems, DAS2000, Suceava, Romania, May 2000, pp. (67-72).

[14] M. V. Micea, D. Chiciudean, L. Muntean, "ECP Standard Parallel Interface for DSP56300 Devices", Application Note AN2085/D Rev. 0, 11/2000, Motorola, Incorporated, USA, 2000.

[15] M. V. Micea, A. Trifu, M. Trifu, "Distributed Digital Signal Processing Using DSP-based Boards and the Khoros Package on LINUX Platforms", in Proceedings of the 5-th International Conference on Development and Application Systems, DAS 2000, Suceava, Romania, May 2000, pp. (61-66).

[16] M. V. Micea, V. Cretu, D. Chiciudean, "Communication Protocols Implementation on DSP-based Systems", in Proceedings of the International Symposium on Systems Theory, SINTES 10, 10th Edition, Automation, Computers, Electronics, Craiova, Romania, May 2000, pp. (C 205-208).

[17] M. V. Micea, "Sisteme de achizitie numerica a datelor: Indrumator de laborator", ("Signal Acquisition and Conditioning Systems: Laboratory Workshops"), Comanda 270/2000, Centrul de multiplicare al Universitatii POLITEHNICA Timisoara, 2000.

[18] V. Cretu, M. V. Micea, I. Guzun, V. Guzun, "Aquarius-DSP, from its Design to Applications Integration", in Transactions on Automatic Control and Computer Science, Vol. 44 (58), No. 1, 2, Periodica Politehnica, Timisoara, Romania, 1999, pp. (5-16).

[19] A. Baya, L. E. Anton, V. Ancusa, M. V. Micea, "Real Time Data Acquisition System for a Compound Straight Pipe", in Scientific and Technical Bulletin of the "Politehnica" University of Timisoara, Mechanics, Tom (44) 58, 1999, pp. (41-50).

[20] M. V. Micea, I. Guzun, V. Guzun, "Performant Multifunction I/O Board for the IBM PC AT", in Proceedings of the International Conference on Microelectronics and Computer Science, Vol. 1, Technical University of Chisinau, Rep. of Moldova, 1997, pp. (167-171).

[21] M. V. Micea, "Real-Time Data Acquisistion and Digital Signal Processing Systems: Present and Prospects", PhD Report No. 1, Department of Computer Science & Engineering, "Politehnica" University of Timisoara, Romania, 2002.

[22] M. V. Micea, "On Design and Implementation of Hard Real-Time Systems for Critical Applications", ("Consideratii referitoare la proiectarea si implementarea sistemelor timp-real stricte pentru aplicatii critice"), PhD Report No. 2, Department of Computer Science & Engineering, "Politehnica" University of Timisoara, Romania, 2003.

[23] M. V. Micea, "Design and Implementation of Hard Real-Time Systems for Critical Applications: the HARETICK system", ("Proiectarea si implementarea sistemelor timp-real stricte pentru aplicatii critice: sistemul HARETICK"), PhD Report No. 3, Department of Computer Science & Engineering, "Politehnica" University of Timisoara, Romania, 2003.