POLITEHNICA University of Timisoara
Department of Computer Science & Engineering
Digital Signal Processing Laboratories – DSPLabs
2, V. Parvan Bv., 1900 – Timisoara, ROMANIA
Tel: +40 256 403271
Web: http://dsplabs.utt.ro

Mihai V. MICEA

Considerații referitoare la proiectarea şi implementarea sistemelor timp-real stricte pentru aplicații critice

Referat de doctorat #2

Conducător ştiințific
Prof. dr. ing. Vladimir CREȚU

2003

Cuprins

1 Introducere

Referatul de față face parte din programul de doctorat cu tema "Contribuții la achiziția şi prelucrarea numerică în timp-real a semnalelor utilizând sisteme de calcul distribuite", sub coordonarea ştiințifică a prof. dr. ing. Vladimir CREȚU.

Sistemele de control digital încorporat (embedded systems) şi achizția şi prelucrarea numerică a semnalelor (DSP-based systems) sunt două domenii înrudite, de interes major în preocupările actuale ale comunității academice şi industriale, fapt dovedit şi de numărul imens de aplicații ce integrează astfel de sisteme în aproape toate domeniile activității umane. O cerință esențială impusă acestor sisteme şi aplicații este operarea în timp-real, cu garantarea respectării termenelor de timp impuse de specificațiile de proiectare şi de mediu. Un număr foarte mare de aplicații au un impact critic asupra mediului înconjurător şi/sau asupra factorului uman cu care interacționează. Exemple de aplicații critice de achiziție, 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 navigație, sistemele de control ale rachetelor, navetelor şi stațiilor spațiale, echipamentul de supraveghere şi control al automobilelor (*automotive*), linii de fabricație 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 consecințe catastrofice asupra mediului înconjurător sau ducând la pierderi de vieți omeneşti.

În cadrul lucrării este abordată problematica sistemelor timp-real (TR) stricte destinate aplicațiilor critice de achiziție şi prelucrare numerică a semnalelor şi de control digital încorporat. Materialul prezentat avansează următoarele idei principale:

- Utilizarea întreruperilor în sistemele TR stricte reprezintă o problemă din punctul de vedere al predictibilității, afectând capacitatea acestora de a garanta în orice condiții de operare respectarea termenelor impuse task-urilor TR stricte;
- Pentru a oferi predictibilitate maximă sistemelor TR stricte destinate aplicațiilor critice de achiziție 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 aplicațiilor TR stricte trebuie să aibă la bază, pentru toate fazele sale, metode şi modele omogene, compatibile.

Capitolul 2 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 atenție deosebită din partea comunității academice şi industriale în ultimele decenii, există încă o serie de

probleme nerezolvate. În Capitolul 3 sunt evidențiate un număr de subiecte care necesită încă soluționare şi care vor fi abordate în capitolele următoare ale lucrării.

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 interacționează sistemele TR şi, utilizând modelul temporal sistem, va fi 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 relații, 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). Tot aici sunt discutate în detaliu aspectele legate de modelarea şi planificarea seturilor de task-uri ce necesită o execuție fixă în cadrul fiecărei perioade a acestora.

Modelul unui sistem complet, OPEN-HARTS, care unifică toate conceptele introduse şi discutate în lucrare, este prezentat în Capitolul 6. Sistemul OPEN-HARTS este conceput ca un set de soluții pentru problemele ridicate de dezvoltarea unitară a unui sistem sau aplicații TR stricte, utilizând modelele de timp sistem, de semnale şi ModX-urile cu planificare şi execuție non-preemptivă.

Concluziile desprinse pe parcursul tratării subiectelor propuse în lucrare, un rezumat al contribuțiilor, şi perspectivele de continuare a cercetării în domeniu, încheie referatul.

Lucrarea aduce o serie de contribuții în domeniul sistemelor TR stricte pentru aplicații critice. O scurtă sinteză a contribuțiilor este prezentată în continuare:

- definirea unui model temporal ce permite specificarea comportamentului în timp al semnalelor şi task-urilor unui STR strict;
- studiul semnalelor şi modelarea comportamentului lor temporal prin introducerea unui set minimal şi complet de parametri;
- 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;
- abordarea problemelor de evaluare a fezabilității setului de task-uri utilizând condiții de necesitate şi/sau suficiență, şi demonstrarea faptului că pentru modelul de task-uri considerat (ModX-uri independente, cu perioadele aliniate la momentul inițial $t_0 = 0$), condiția a doua de necesitate enunțată în [Jeffay 91] nu este aplicabilă;
- discutarea cazurilor, mai apropiate de realitate, ale sistemelor TR stricte ce execută, pe lângă setul de ModX-uri ale aplicației, şi un task special – planificatorul online, ce utilizează o tabelă de dispatch de dimensiuni fixe, bine determinate, pentru planificarea ModX-urilor în cicluri cu număr constant de execuții;
- abordarea problemei modelării şi planificării seturilor de task-uri ce necesită o execuție fixă în cadrul fiecărei perioade a acestora;

- introducerea noțiunii de ModX cu execuție fixă (FModX) şi demonstrarea unui model matematic al acestuia, bazat pe aritmetica modulară şi pe funcția treaptă unitate;
- studierea planificării non-preemptive a seturilor compuse din două FModXuri independente, pe baza funcției de mapare şi a proprietăților demonstrate ale acesteia;
- introducerea şi demonstrarea condițiilor de necesitate şi suficiență ale planificabilității non-preemptive a seturilor de două FModX-uri;
- 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 aplicațiilor 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 execuție aplicațiilor TR critice).

2 Sisteme timp-real şi control digital încorporat: concepte actuale

Pe măsură ce sistemele digitale de calcul şi control sunt integrate în tot mai multe medii şi aplicații din toate domeniile de activitate umane, a crescut importanța şi interesul acordat sistemelor de achiziție 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 APNS şi de control digital încorporat sunt în majoritatea cazurilor utilizate în aplicații cu cerințe de operare în timp real şi, în multe cazuri, implementează funcții critice de procesare şi control.

Capitolul de față reprezintă o scurtă trecere în revistă a unor concepte actuale legate de sistemele şi aplicațiile timp-real, care vor fi abordate apoi în restul lucrării.

2.1 Sisteme timp-real: definire şi caracteristici generale

Oxford Dictionary of Computing dă următoarea definiție a unui sistem timp-real [Ostroff 92]:

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

[Young 82] defineşte un STR ca fiind: "Orice activitate sau sistem de procesare a informației, care trebuie să răspundă după o perioadă finită şi bine specificată, unor stimuli de intrare generați 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, Ghosh 94, Panzieri 93, Parashkevov 94, Stankovic 92a, Stankovic 92b].

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

- *Sisteme transformaționale*: sunt sistemele care procesează un set de date de intrare pentru a obține 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*: interacționează 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*, acestea reacționează 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 reacționeze cu succes în ritmul dictat de mediul înconjurător. Exemple: controloarele de proces, sistemele de control digital încorporat, sistemele de achiziție şi prelucrare numerică a semnalelor, etc.

Sistemele de calcul şi aplicațiile reale nu aparțin strict doar uneia din categoriile de mai sus. Astfel, sistemele TR sunt, în mod uzual, sisteme reactive, dar care permit un anumit nivel de interacțiune, cu operatorul de exemplu. Figura 2-1 prezintă modelul de principiu al STR, în care sunt evidențiate sistemul de control (STR), sistemul controlat (mediul înconjurător) şi partea de interacțiune cu operatorul uman.

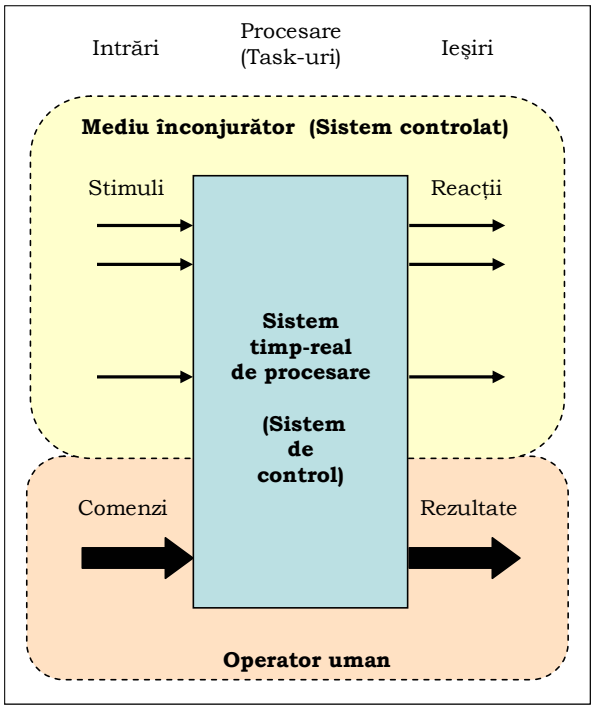

Figura 2-1. Schema de principiu a STR

Mediul în care operează un anumit STR are un rol extrem de important în conceperea şi proiectarea acestuia [Stankovic 92b]. O pondere majoră a specificațiilor STR derivă din restricțiile pentru semnalele de intrare/ieşire impuse sistemului de către mediul înconjurător. De exemplu, perioada de apariție a unui anumit semnal de intrare are implicații directe asupra intervalului maxim de timp pe care STR îl are la dispoziție 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 față.

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 secvență de intrări va produce de fiecare dată o aceeaşi secvență de ieşiri [Berry 00]. Determinismul este o caracteristică de bază a sistemelor reactive şi unul din criteriile care le deosebeşte față de sistemele interactive. Identificarea (sau modelarea) unui mediu determinist în care va opera un STR, stă la baza specificării şi proiectării acesuia.

Cerințele 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 aplicațiilor TR o coordonată esențială, cu impact în toate fazele dezvoltării şi exploatării acestora, de la specificare a întregului sistem şi până la execuția la nivel de task. Cerințele 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 informație obținută de la senzori sau cele de acționări asupra mediului sunt de tip periodic şi cu cerințe de timp severe.

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

- *STR critice*: includ funcții (task-uri) pentru care respectarea termenelor de execuție este o problemă critică. Exemple de astfel de task-uri sunt cele de monitorizare a temperaturii reactorului într-o centrală nucleară, sistemele de navigație ale unei rachete de croazieră sau sistemul tip "fly-by-wire" de control al zborului avioanelor moderne. Nerespectarea termenelor specificate pentru aceste sisteme are consecințe catastrofale pentru mediu, pierderi de vieți omenești, etc.
- *STR stricte*: includ task-uri pentru care depăşirea termenelor de execuție implică o comportare eronată a sistemului. Cu alte cuvinte, valoarea de execuție a task-urilor TR stricte este anulată în cazul în care execuția acestora depăşeşte termenele specificate. Nerespectarea termenelor în STR stricte are efecte destul de grave asupra aplicațiilor ce le utilizează (avarierea sau distrugerea de echipamente şi bunuri materiale, apariția de erori în ciclul de producție, anularea de diverse tranzacții, etc.).
- *STR lejere*: sunt sisteme ce conțin numai funcții (task-uri) a căror valoare de execuție nu se anulează, ci se diminuează doar, odată cu depăşirea termenelor specificate. Exemple de astfel de sisteme se găsesc în domeniul multimedia şi al comunicațiilor uzuale. În cazul unei video-conferințe, pierderea unui cadru de imagine la recepție nu este o problemă catastrofală, comunicația desfăşurându-se în continuare. Bineînțeles însă că valoarea operării întregului sistem este cu atât mai mare cu cât se pierd mai puține cadre video.

Tehnicile şi metodele implicate în cazul task-urilor TR stricte (critice) diferă foare mult față 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 prealocate (incluzând rezervarea resurselor necesare, pentru intervalele de timp corespunzătoare execuției lor) şi pre-planificate, având ca efect garantarea respectării termenelor de execuție 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 performanța

în situațiile de operare medie (încărcare medie a sistemului, situații uzuale de operare, etc.). Aşa cum este evidențiat 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 fiabilității trebuie luată în considerare ca un element esențial 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 fezabilității 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 apariția 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 execuție a task-urilor, pentru situația cea mai defavorabilă de operare*.

2.2 Timpul ca şi coordonată esențială a specificării şi operării STR

Cercetările din ultimii peste 25 de ani în domeniul sistemelor TR subliniază necesitatea introducerii coordonatei temporale în toate fazele dezvoltării STR: specificare, verificare/validare, programare, analiză, planificare şi execuție.

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 cerințelor şi de specificare a STR, şi se concentrează mai degrabă asupra modelării comportării sistemelor decât asupra aspectelor structurale şi funcționale ale acestora.

Comportarea unui sistem se referă la modul în care acesta reacționează 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 *funcționale* se referă la transformările informației în cadrul sistemului (procesarea informației).

Logicile temporale sunt definite ca structuri de forma:

$$\langle \Im, R, V \rangle \tag{2-1}$$

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

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

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

Pentru logicile temporale, relația R este denumită *relația de precedență* şi e notată cu "<". De regulă, "<" este o relație tranzitivă şi nereflexivă, realizând o ordonare parțială a domeniului temporal, $\Im$ [Bellini 00]. Proprietățile pe care relația "<" 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 instanțe de timp există întotdeauna o a treia. Se observă că există proprietăți care se exclud reciproc, cum sunt de exemplu, Proprietatea 2-8 cu Proprietatea 2-9. Fiecare din aceste proprietăți 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 instanță 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 relațiilor dintre evenimente în termeni de instanțe de timp (puncte din domeniul temporal). Aceste logici definesc intervalele ca seturi conectate de puncte succesive. Exprimarea relațiilor 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 instanță de timp (un punct din domeniul temporal) este reprezentată ca un interval de dimensiune (durată) unitară. Relațiile dintre intervale sunt exprimate mai uşor şi mai intuitiv în aceste modele, utilizând operatori specifici comparativi (exprimă relațiile dintre intervale) şi combinatorici (exprimă modul de combinare al intervalelor), sau operatori care precizează limitele intervalelor pe baza valorilor de adevăr ale predicatelor. Figura 2-2 ilustrează relațiile posibile între două intervale, împreună cu operatorii comparativi aferenți, conform modelului temporal bazat pe intervale propus în [Allen 83] şi [Allen 94].

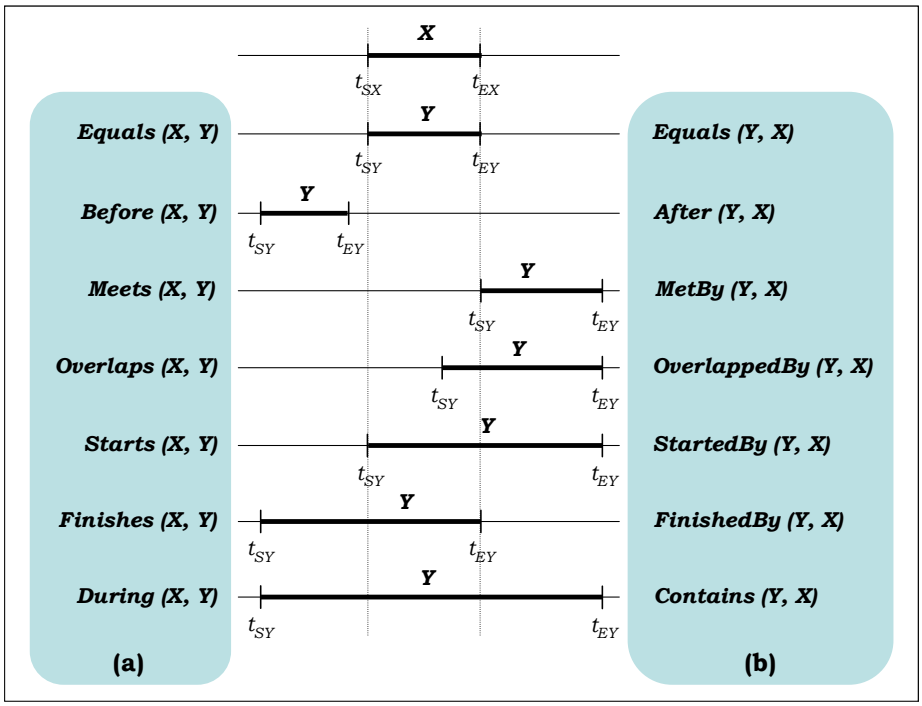

Figura 2-2. Relațiile posibile, directe (a) şi reciproce (b), între două intervale temporale

Operatorii utilizați în modelul temporal ilustrat în Figura 2-2 pot fi transformați 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,

$$\Box_{[2,9]}\, A \tag{2-1}$$

semnifică faptul că formula (propoziția) A este adevărată pentru toate instanțele 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 condiții, operatorul *Before*(*X*,*Y*) din Figura 2-2 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 relații temporale cantitative, cum ar fi distanța dintre două evenimente sau durata unor evenimente, exprimate în unități de timp. *Posibilitatea exprimării de constrângeri temporale cantitative este esențială pentru specifiarea STR*.

Timpul poate fi modelat în manieră *implicită* sau *explicită*, şi *relativă* sau *absolută*. Un model temporal este implicit atunci când semnificația formulelor depinde de evaluarea momentului curent al timpului, acesta fiind considerat implicit în formule. De exemplu, formula (2-1) este echivalentă cu:

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

unde t_0 este evaluarea timpului (instanța 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 proprietăți 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 instanțele pare de timp [Bellini 00].

Referirea timpului este considerată absolută când valoarea timpului curent este determinată în funcție 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 instanțelor de timp sunt exprimate direct în secunde sau milisecunde, etc. (adică au valori absolute, corespunzătoare ceasului referit). Modelele temporale relative exprimă duratele şi instanțele de timp în unități temporale, relativ la o instanță a timpului absolut, exprimată în secunde, milisecunde, etc. Determinarea valorii absolute a elementelor temporale pentru modelul relativ, este amânată pentru faza implementării sistemului.

2.3 Tehnici de planificare

Rolul algoritmilor de planificare în cadrul STR este găsirea de soluții fiabile pentru asignarea în timp a resursei principale a sistemului – procesorul (sau procesoarele), către task-urile sistemului, astfel încât să nu existe suprapuneri ale execuțiilor 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 aplicațiilor 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 funcție 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 execuției unui task nu sunt admise întreruperi).

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

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

Algoritmii de planificare statică presupun existența unei analize offline a planificabilității aplicației ce urmează a fi executată pe STR. Rezultatul analizei constă într-o tabelă care specifică planificarea şi execuția fiecărui task în parte, împreună cu timpii lor de execuție, sau este stabilită (prin asignarea de priorități, de exemplu) ordinea în care executivul STR va lansa în execuție task-urile aplicației. Cei mai frecvenți algoritmi de planificare statică utilizați în cadrul sistemelor TR sunt algorimii de *planificare ciclică* [Ramamritham 94, Ghosh 94] şi algoritmul *RMS* (*Rate Monotonic Scheduler*) [Liu 73, Kim 80, Audsley 91]. De exemplu, RMS

asignează priorități distince pentru fiecare task al aplicației, în manieră monoton descrescătoare a frecvenței de execuție a acestora, astfel încât prioritarea mai mare este atribuită task-ului cu frecvența mai mare şi aşa mai departe.

Avantajul tehnicilor statice de planificare constă în obținerea unei predictibilități mari a operării sistemului, cât şi în complexitatea redusă a algoritmilor, facilitând astfel implementarea acestora pe arhitecturi simple, destinate unor aplicații 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 condiții de operare sau de mediu, implică reluarea analizei offline şi recalcularea întregii planificări), utilizarea procesor relativ mică a aplicațiilor pe care algoritmii sunt capabili a le planifica şi fenomenul de inversare a priorităților (*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 concepuți şi studiați algoritmii de planificare dinamică. Aceşti algoritmi asignează în mod dinamic priorităţile task-urilor aplicației, 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 eficienți 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 situații tranzitorii de supraîncărcare a sistemului.

EDF asignează în mod dinamic priorități task-urilor aplicației după criteriul celui mai apropiat termen de execuție (*deadline*): task-ul cu cel mai apropiat termen primeşte prioritatea cea mai mare. *MLF*, reprezintă o versiune modificată a lui *EDF*, în care se ține cont şi de durata de execuție a task-urilor, nu numai de termenele de execuție. *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 execuție, considerând durata maximă de execuție 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 îmbunătățească predictibilitatea mai ales în situațiile tranzitorii de supraîncărcare a sistemului, este *MUF*. Algoritmul prevede pentru fiecare task, asignarea (statică) a unui parametru ce specifică în mod explicit gradul de urgență, şi determinarea dinamică a unei priorități care este invers proporțională cu laxitatea task-ului (ca în cazul *MLF*). Gradul de urgență al task-urilor este o combinație a două priorități fixe: (a) nivelul critic, cu precedență mai mare decât prioritatea dinamică a task-ului, şi (b) prioritatea utilizator, cu precedență mai mică decât prioritatea dinamică. Prin această combinație, se încearcă ajutarea algoritmilor de planificare în diferențierea dintre task-uri mai importante şi mai puțin importante, prin noțiunea 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 atenția 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 complexității algoritmilor (de ordin nepolinomial, NP [Rădulescu 02]), cât şi al resurselor de calcul (timp) necesare. În consecință, planificarea task-urilor într-un sistem distribuit nu mai poate aplica seturi de criterii bine definite, în operații de căutare exthaustivă pentru selecția task-urilor ce vor fi planificate, ci sunt necesare abordări euristice, probabilistice şi de aproximare.

Au fost propuse diverse metode pentru măsurarea şi compararea performanțelor 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 performanței algoritmilor de planificare, cât şi pentru derivarea condițiilor de fezabilitate a planificării (vezi şi Capitolul 5):

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

unde: C_i reprezintă durata de execuție 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 planificabilității* (*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:

$$SB_A \left(= BU_A\right) = \max_{\forall \boldsymbol{M}} \left\{ U^{\boldsymbol{M}} \mid \boldsymbol{M} \text{ e fezabil cu } A \right\} \tag{2-3}$$

unde A este algoritmul de planificare evaluat.

Evaluarea algoritmilor de planificare cu ajutorul limitei superioare a planificabilității exprimată de (2-3), 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 distribuție 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 planificabilității unitară (100%) pentru toate seturile generice de task-uri (SB_{EDF} = 1.00), dar, aşa cum a fost menționat anterior, algoritmul nu poate garanta respectarea termenelor tuturor task-urilor pentru cazurile de supraîncărcare tranzitorie a sistemului.

Alte exemple de parametri destinați evaluării performanțelor 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 execuție şi până se termină execuția lui, şi timpul procesor consumat pentru execuția efectivă a task-ului. Cu cât *NMRT* este mai mare, cu atât este mai mare timpul "de inactivitate" al task-urilor, eficiența algoritmului fiind deci mai scăzută.

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

Din punctul de vedere al sistemelor TR stricte, destinate aplicațiilor critice, predictibilitatea operării este o cerință esențială. 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 potențialul garantării execuției task-urilor cu respectarea termenelor specificate, chiar şi în condițiile 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 față vom aborda problema planificării non-preemptive a seturilor de task-uri modelate special pentru sisteme TR stricte.

3 Problemele dezvoltării STR stricte pentru aplicații critice

3.1 Dezvoltarea STR – concepte şi arhitecturi

Marea majoritate a lucrărilor din domeniul sistemelor TR evidențiază faptul că dezvoltarea sistemelor şi aplicațiilor 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) Concepție şi proiectare
2) Specificarea funcțională (programare) şi de comportare în timp
3) Verificare/validare formală
4) Analiza temporală de program şi de fezabilitate a execuției
5) Generarea codului, a planificării şi implementarea pe o arhitectură țintă
6) Testarea implementării

Toate etapele dezvoltării STR trebuie să aibă în vedere introducerea timpului ca şi coordonată esențială a sistemului, şi nu doar ca o caracteristică în funcție de care să se adapteze sistemul gata dezvoltat [Stankovic 92b]. Tot în această lucrare este evidențiată dificultatea cerințelor 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 aplicațiile vizate şi particularizate pentru arhitecturile implementării. Multe sisteme TR se bazează pe adaptarea şi optimizarea (în sensul creşterii vitezei de reacție) a unor versiuni de sisteme de operare time-sharing tradiționale. 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 execuția proceselor ținând cont de unele specificații de timp.

Sistemele de operare tradiționale se bazează pe noțiunea că task-urile aplicației 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 situațiile comune de operare ("average case behavior").

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

- sunt vizate schimbările mai rapide de context;
- nucleul va avea dimensiuni mai mici (cu funcționalitatea 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 secvențiale speciale, capabile a acumula date într-un ritm rapid.

Pentru a face față cerințelor temporale impuse de operarea TR, nucleul unui sistem tradițional va fi adaptat astfel:

- gestionează un ceas de timp-real (*RTC – Real-Time Clock*);
- se oferă un mecanism de planificare bazat pe priorități;
- sunt puse la dispoziția task-urilor aplicațiilor 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 execuția.

Din punct de vedere al arhitecturilor sistem, în prezent există o multitudine de mecanisme şi concepte special prevăzute pentru a creşte eficiența 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 execuție (*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 menționate mai sus induc o comportare puternic asincronă şi impredictibilă din punct de vedere al timpilor de execuție pentru sistemele TR, şi trebuie în consecință, evitate în faza de proiectare a arhitecturii hardware a STR.

3.2 Predictibilitatea execuției task-urilor TR cu termene stricte

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

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

Predictibilitatea are implicații directe în conceperea mecanismelor de alocare a resurselor sistem (procesor, memorie, accese I/O, comunicații, etc.), impunând necesitatea integrării restricțiilor de timp [Stankovic 92b]. În particular, proiectarea algoritmilor de planificare pentru sistemele TR, trebuie să aibă în vedere asigurarea predictibilității de operare, cel puțin pentru task-urile TR cu specificații 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 planificabilității (sau, o verificare a fezabilității) sistemului. Pentru cazul sistemelor TR stricte, unde respectarea specificațiilor temporale este o cerință esențială, ce trebuie garantată în orice condiții de operare, faza de analiză a planificabilității trebuie efectuată înaintea lansării în execuție a sistemului (aplicației).

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

- Întreruperile sunt cea mai frecventă cauză a fenomenului de inversare a priorităților (*priority inversion*) dintr-un sistem cu planificare pe bază de priorități: execuția 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 apariției unor evenimente asincrone în sistem.
- Rutinele de tratare a întreruperilor sunt executate în contexte diferite de cele ale task-urilor aplicației, forțând astfel utilizarea variabilelor globale pentru a putea schimba informație cu restul aplicației.
- Din punct de vedere al testării aplicațiilor, 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 al programului în interiorul handlerelor, iar pe de altă parte, din cauza asincronismului apariției întreruperilor.

Se impune în consecință, mai ales în cazul sistemleor 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 conține o discuție şi mai detaliată a avantajelor şi dezavantajelor celor două abordări – preemptivă şi non-preemptivă. Tot aici va fi prezentată o clasă de aplicații TR stricte, ce conțin task-uri cu execuție fixă în cadrul perioadei şi a căror planificare nu poate fi rezolvată cu algoritmi preemptivi.

3.3 Specificarea, programarea şi analiza temporală a aplicațiilor TR critice

În prezent, există o varietate mare de metode de specificare şi verificare formală a sistemelor şi aplicațiilor 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 92, Parashkevov 94]: Time Petri Nets, Timed Petri Nets (bazate pe rețelele 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 posibilități 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 verificăii automatizate a proiectelor sistemelor TR.

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

- dependența cât mai redusă față de hardware şi sistemul de operare a platformelor țintă;
- trebuie să posede facilitățile limbajelor moderne de programare (set puternic de tipuri de date, programare structurată, modularitate, interfețe 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 concurenței, a comunicării şi sincronizării proceselor aplicației;
- trebuie să ofere programatorului mecanismele necesare descrierii explicite a proprietăților şi constrângerilor temporale ale codului programelor: execuție periodică, contoare de timp, termene pentru task-uri şi grupuri de task-uri, întârzieri, etc.;
- restricționarea sau chiar interzicerea construcțiilor de program şi a structurilor de date ce nu sunt limitate în timp şi spațiu: recursivitatea, structuri de memorie dinamică, bucle cu un număr nelimitat sau necunoscut de iterații la momentul compilării, etc.;
- trebuie să ofere informații suplimentare utilizate în analiza planificabilității codului compilat.

3.4 Modelul unui STR strict pentru aplicații critice

Sintetizând cerințele 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 aplicații critice de achiziție 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 fezabilității.
- Timpul, ca şi coordonată esențială 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 predictibilității execuției task-urilor TR stricte ale sistemului reprezintă o cerință esențială, ce afectează toate etapele dezvoltării şi implementării.
- Algoritmii de planificare trebuie atent concepuți astfel încât să asigure execuția predictibilă, cu respectarea termenelor impuse task-urilor TR stricte şi conferind sistemului, pe de altă parte, eficiență şi flexibilitate cât mai mari.
- Programarea task-urilor aplicațiilor 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 predictibilității, 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 atenție 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.

4 Informația de timp, evenimente şi acțiuni. Modelul task-ului TR strict

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

Capitolul de față analizează informația 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 uşurință 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

Funcționarea sistemelor de calcul în general, şi a STR în particular, se bazează pe o frecvență de tact generată de un oscilator intern sau extern. Frecvența de tact defineşte aşa-numitul "ciclu procesor" sau "ciclu instrucțiune" al unității de procesare a sistemului, reprezentând durata execuției unei instrucțiuni procesor (exprimată în secunde, Hz sau perioade de tact). În acelaşi timp, frecvența 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, care, pentru o aplicație oarecare, operează la o frecvență 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 inițializarea contorului RTC), fiind denumit "momentul inițial".

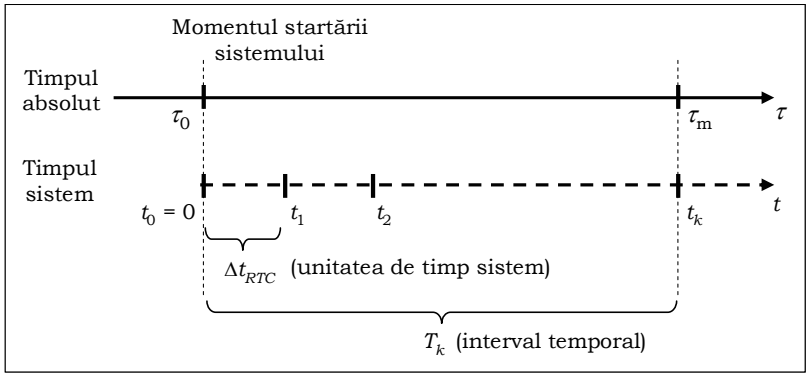

Figura 4-1. Timpul sistem şi timpul absolut pentru STR

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 instanțe succesive ale timpului sistem, t_i şi t_{i+1} sunt distanțate 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 unității temporale sistem şi faptul că aceasta permite existența 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 relația de corespondență a unei instanțe de timp (un punct din cele două domenii):

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

Importanța relației (4-2) şi a utilizării ambelor modele temporale derivă din faptul că în faza de specificare a comportamentului temporal al STR, utilizatorul/programatorul de aplicații 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/aplicației TR, anume în faza de verificare şi validare, este necesară convertirea în mod corespunzător a specificațiilor, 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: • $t \in \mathrm{N}$, pentru modelul temporal sistem, • $t \in \mathrm{R}$, 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 aplicații TR, etc.) clasifică anumite tipare semnificative de schimbare [Allen 94]. Există destul de puține restricții în stabilirea semanticii evenimentelor, cu excepția că acestea trebuie să implice cel puțin un obiect într-un interval de timp sau cel puțin o schimbare de stare. Diferența 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 accepțiune mai restrânsă şi din punctul de vedere al specificării şi operării STR prezentat în lucrarea de față, 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 apariții:
  b.1) semnale cu apariție singulară
  b.2) semnale cu apariție temporară
  b.3) semnale cu apariție 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 apariție a lui S_i are loc la momentul t_k, atunci cea de-a (k+1) apariție 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 comunicațiile sincrone, liniile de date şi control din sistemele de achiziție a datelor, etc.

Exemplul 4-2.

Considerăm un sistem de achiziții de date ce include o schemă de conversie analog-numerică, CAN, (Figura 4-2) şi care este proiectat pentru achiziționarea de date la o rată constantă de eşantionare F_e = 1 KHz. Subsistemul operează în regim comandat, necesitând activarea continuă şi cu o perioadă bine determinată a liniei de lansare a unui ciclu de conversie (linia "STC").

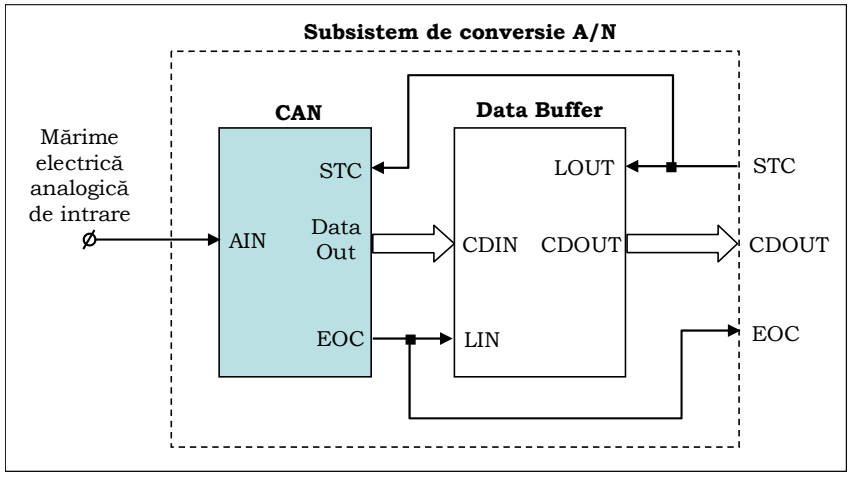

Figura 4-2. Subsistemul de conversie A/N

Figura 4-2 prezintă o arhitectură simplă de conversie A/N cu operare controlată din exterior (subsistem tip "slave") [Micea 00a, Micea 00b]. Sistemul care comandă operarea CAN (sistemul "master") activează periodic linia "STC" ("Start Conversion"), care are dublu efect:

i) lansează un nou ciclu de conversie pentru circuitul CAN,

ii) comandă (activează linia "LOUT" – "Latch Out") eliberarea datelor reținute de registrul tampon ("Data Buffer") spre magistrala de date de ieşire ("CDOUT" – "Conversion Data Out").

În momentul în care linia "STC" a circuitului CAN este activată, începe un nou ciclu de conversie. La terminarea acestuia, rezultatele conversiei sunt disponibile pe liniile "Data Out", iar linia "EOC" este activată permițând stocarea rezultatelor în registrul tampon (activarea liniei "LIN" – "Latch In").

Rata de achiziție a datelor, F_e, implică faptul că semnalele "STC" şi "CDOUT" trebuie să fie periodice, cu perioadele:

$$T_{pr}^{STC} = T_{pr}^{CDOUT} = \frac{1}{F_e} = 1 \text{ [msec]}$$

Ca observație, semnalul "EOC" poate fi la rândul său periodic, însă doar în condițiile în care ciclurile de conversie ale CAN sunt egale ca durată (ceea ce în realitate nu se întâmplă, de fapt).

❑

a.2) *Semnale aperiodice (sporadice)*

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

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

Se observă că relațiile (4-4) şi (4-5), ce specifică momentul unei noi apariții a semnalului periodic, respectiv aperiodic, S_i, față de instanța ultimei apariții, sunt asemănătoare. Această observație permite abordarea (tratarea) celor două tipuri de semnale într-o manieră unitară, aşa cum se va vedea în secțiunea următoare.

Exemple de semnale aperiodice sunt comunicațiile asincrone, interfețele cu operatorul, subsistemele de comandă cu bucle de reacție şi reglare, interfețe intrare-ieşire ce operează pe bază de întreruperi, etc.

Exemplul 4-3.

Într-o aplicație de comandă a unui motor electric cu buclă de reacție, sistemul digital de control va activa o linie care reduce tensiunea aplicată motorului de câte ori algoritmul său de determinare a vitezei de rotație a motorului detectează depăşirea vitezei nominale. Evident, activarea acestui semnal nu este periodică şi depinde de condițiile concrete de operare ale motorului, tensiunea de alimentare, temperatură, etc. Cu toate acestea, se poate determina un interval minim de timp ce separă aparițiile consecutive ale semnalului şi care este cel puțin egal cu durata execuției algoritmului ce implementează bucla de reglare.

❑

b.1) *Semnale cu apariție singulară*

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

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

b.2) *Semnale cu apariție temporară*

Semnalele cu apariție temporară caracterizează anumite procese ce se desfăşoară pe un interval de timp limitat de-a lungul operării sistemelor şi a interacțiunii

acestora cu mediul. Faptul că un anumit semnal S_i are apariție temporară poate fi caracterizat prin durata intervalului temporal în care au loc evenimentele apariției semnalului (durata intervalului în care semnalul este "activ", $T_{act}^{S_i}$). În cazul în care semnalul este şi periodic, durata intervalului de apariții poate fi echivalată cu numărul de apariții:

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

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

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

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

Exemplul 4-4.

În cazul unei macarale cu deplasare controlată digital prin intermediul unei interfețe cu operatorul de tip "joystick", este posibilă specificarea celor patru direcții de deplasare (stânga, dreapta, înainte şi înapoi) prin poziționarea corespunzătoare a manetei. Durata deplasării (cu viteză constantă) depinde de durata acționării manetei în poziția stângă. Semnalele periodice generate de joystick, ce corespund fiecărei poziții a manetei, sunt cu apariție temporară.

În cazul acestui sistem însă, nu se poate specifica apriori durata intervalurilor temporale în care apar semnalele generate de joystick. Pe de altă parte, se poate specifica o limită pentru activările fiecărui semnal (în număr de apariții, sau ca durată a intervalului temporal), pentru a se evita depăşirea unor zone de siguranță în operarea macaralei.

❑

b.3) *Semnale cu apariție permanentă*

Semnalele cu apariție 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 apariție temporară, în condițiile în care numărul de apariții şi durata intervalului de apariții sunt infinite.

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

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

Figura 4-3 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).

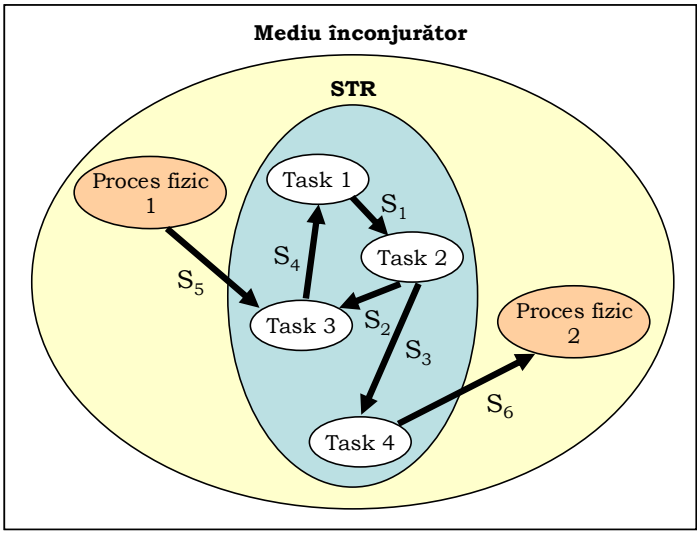

Figura 4-3. Ilustrare a celor trei tipuri de semnale: de intrare, de ieşire şi interne

c.1) *Semnale de intrare*

Semnalele de intrare au ca sursă mediul înconjurător în care (asupra căruia) operează STR. Față de aceste semnale, STR se comportă ca un sistem reactiv: apariția 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 reacția STR.

Tratarea semnalelor de intrare este o poblemă deosebit de importantă pentru dezvoltarea şi operarea STR stricte, şi modul în care este rezolvată diferențiază diferitele arhitecturi şi concepte de sisteme TR. Problema va fi abordată în detaliu în secțiunea următoare.

c.2) *Semnale de ieşire*

Semnalele de ieşire sunt semnalele prin intermediul cărora STR acționează 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 interacțiunii dintre task-urile sistemului şi constau de regulă din: semnale de sincronizare şi de notificare, şi semnale ce conțin informație schimbată între task-uri (comunicația 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, discutați în această secțiune. Parametrii din tabel vor fi utilizați pentru modelarea semnalelor în faza de specificare a STR, urmând a avea implicații 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 apariții consecutive ale semnalului S_i.	Semnale aperiodice (sporadice)
$T_{act}^{S_i}$	Durata intervalului în care au loc apariții ale semnalului S_i.	Semnale temporare
N^{S_i}	Numărul de apariții 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: acțiuni TR stricte

Într-o accepțiune generică, acțiunile sunt definite ca activități pe care un sistem (obiect, persoană, etc.) le desfăşoară în anumite condiții [Allen 94]. Din perspectiva tematicii abordate în lucrarea de față, acțiunile sunt văzute în strânsă relație de cauzalitate cu evenimentele. Cu alte cuvinte, apariția unui anume eveniment sau a unei secvențe bine definite de evenimente, provoacă o acțiune, şi reciproc, o acțiune anume, provoacă apariția unui eveniment sau a unui set bine precizat de evenimente. Relația de cauzalitate eveniment ↔ acțiune este în strânsă concordanță cu accepțiunea 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 acțiune este reprezentată printr-un task sau o secvență bine definită de task-uri specifice (subgraf de task-uri). Relația de cauzalitate dintre evenimente (semnale) şi acțiuni (task-uri) implică faptul că unele proprietăți ale semnalelor determină proprietăți 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. Detalierea acestui model se va face pe parcursul secțiunii de față şi în secțiunea următoare, în care se propune un model temporal minimal, complet, al task-ului TR strict.

Definiția 4-1.

Un *task* (notat M_i) reprezintă o secvență de instrucțiuni pe care sistemul o execută în scopul desfăşurării unor acțiuni. Execuția task-urilor în cadrul sistemelor TR se face într-o manieră *planificată*, ținând 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 obține apariția 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 execuții. Notăm cu N^{M_i} numărul total de execuții ale task-ului M_i ($N^{M_i} \geq 1$). Fiecare ciclu k de execuție începe din momentul invocării $t_{rq,k}^{M_i}$, şi se termină la momentul invocării pentru ciclul următor, $t_{rq,k+1}^{M_i}$.

Definiția 4-2.

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

❑

Aşa cum rezultă din Capitolele 2 şi 3 ale lucrării, durata de execuție a unui task variază de la o execuție la alta. Aşadar, $T_{ex,k}^{M_i}$ nu este constantă, depinzând de o serie de factori, cum ar fi valoarea momentană a informației preluată de la task-urile executate anterior şi de calea pe care o urmează execuția în cadrul secvențelor de instrucțiuni ale task-ului (de exemplu, în cazul instrucțiunilor de ramificare de tip "for" sau "case"), etc. În cazul sistemelor TR şi în particular, al STR stricte, faptul că durata de execuție a task-urilor nu e constantă de la o execuție 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 secțiunile şi capitolele următoare ale lucrării.

Definiția 4-3.

Numim *perioadă* a task-ului M_i (*period*), notată cu $T_{pr}^{M_i}$, durata minimă a unui ciclu de execuție 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 execuție.

❑

Se observă că Definiția 4-3 şi relația (4-6) permit atât existența 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 (relațiile (4-4), (4-5) şi [Jeffay 91]).

Definiția 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 execuție 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 execuție 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 Definiția 4-4, spunem că task-ul periodic M_i solicită $T_{ex,k}^{M_i}$ unități din timpul procesor în fiecare interval $[t_{rq,k}^{M_i}, t_{rq,k}^{M_i} + T_{pr}^{M_i}]$.

Definiția 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:

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

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

❑

Un sistem poate conține un număr oarecare de task-uri, ce pot fi *independente* sau *dependente* unul față de celălalt. Dependențele dintre task-urile unui STR pot fi de două tipuri:

- *dependență de control* (*control dependence, control flow*), dacă secvența de procesare (programul) specifică faptul că un task M_j succede în mod obligatoriu altui task M_i, şi nu invers. În general, dependența de control dintre task-uri este specificată în faza de programare a aplicației, prin intermediul unor mecanisme ca: apelare de task-uri, sincronizare inter-task, etc.;
- *dependență de date* (*data dependence, data flow*), dacă informația procesată de un task M_j (parametrii de intrare, variabilele interne) depinde de informația procesată în prealabil de un task M_i (variabilele interne, parametrii de ieşire). De regulă, dependența de date dintre task-uri este specificată în faza de programare a aplicației, prin intermediul unor mecanisme ca: transmiterea de parametri de intrare/ieşire, variabile globale, etc.

Din punct de vedere formal, cele două tipuri de dependență inter-task pot fi grupate cu ajutorul aşa-numitei *dependențe de program* (*program dependence*). Dependențele de program ale task-urilor unei aplicații impun restricții (de multe ori, dificil de rezolvat) pentru planificarea în execuție a acestora. Un task M_j nu poate fi planificat şi executat la un moment oarecare t decât în condițiile în care sunt îndeplinite *toate* dependențele sale de program, semnificând de exemplu, ca task-ul M_i să fi obținut prin execuție, până la momentul $(t - 1)$, valorile parametrilor săi de ieşire din care o parte sunt transmişi lui M_j ca parametri de intrare.

Definiția 4-6.

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

❑

Sistemele TR stricte impun termene limită pentru execuția task-urilor. În cazul task-urilor TR stricte, depăşirea acestor termene implică funcționarea eronată a sistemului, uneori cu consecințe 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.

❑

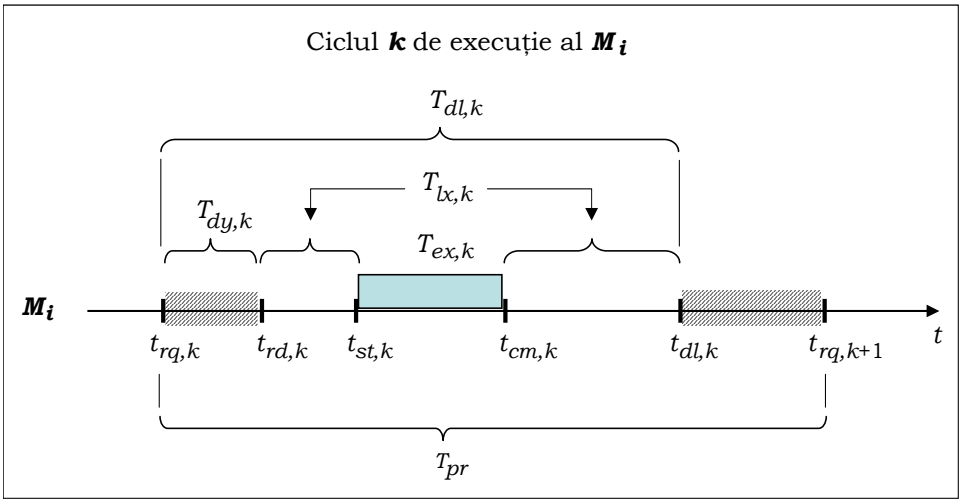

Figura 4-4. Modelul temporal al task-ului TR strict, M_i

Figura 4-4 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	Semnificație
$t_{rq,k}^{M_i}$	*Momentul de invocare (Request time)*	Momentul în care M_i este invocat de STR. Defineşte începutul ciclului k de execuție a lui M_i.
$t_{rd,k}^{M_i}$	*Momentul gata de execuție (Ready time)*	Momentul de la care M_i poate fi planificat pentru execuție.
$t_{st,k}^{M_i}$	*Momentul de start (Start time)*	Momentul în care M_i îşi începe efectiv execuția.
$t_{cm,k}^{M_i}$	*Momentul de terminare (Completion time)*	Momentul în care M_i îşi termină execuția şi este eliberat procesorul sistemului.
$t_{dl,k}^{M_i}$	*Termenul limită (Deadline)*	Momentul până la care execuția lui M_i are valoare în cadrul sistemului.
$T_{ex,k}^{M_i}$	*Durata de execuție (Execution interval)*	Intervalul de timp necesar execuției lui M_i în cadrul sistemului.
$T_{dl,k}^{M_i}$	*Intervalul limită (Deadline interval)*	Intervalul de timp în care execuția lui M_i poate fi planificată pentru o funcționare corectă a sistemului.
$T_{dy,k}^{M_i}$	*Intervalul de întârziere (Delay interval)*	Intervalul de timp după care execuția 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 execuția lui să nu depăşească termenul limită.
$T_{pr}^{M_i}$	*Perioada (Period)*	Durata minimă a unui ciclu de execuție a lui M_i.
N^{M_i}	*Numărul de execuții (Execution count)*	Numărul total de execuții ale lui M_i (numărul de cicluri de execuție).

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

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

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

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

* * *

Urmând clasificarea semnalelor, detaliată în secțiunea precedentă, 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, relația de cauzalitate este de tipul "acțiune → eveniment". Comportamentul în domeniul timp al semnalelor de ieşire este complet determinat de proprietățile temporale ale task-urilor care le generează. Prin urmare, specificațiile 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} execuții, 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 apariții.

În faza de specificare/verificare a STR este necesară punerea în corespondență 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 comunicației dintre task-urile STR va fi tratată separat, în secțiunea următoare, ce descrie 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. Relația de cauzalitate în acest caz este de tipul "eveniment → acțiune", comportamentul în timp al task-urilor ce tratează semnalele de intrare trebuind să corespundă specificațiilor 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 secțiunea anterioară (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.

Definiția 4-8.

Definim *intervalul de răspuns* pentru semnalul S_i, notat cu $T_{rs}^{S_i}$, intervalul de timp inițiat de momentul apariției 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 execuția de task-uri aperiodice, invocate de întreruperi. Existența în cadrul STR a task-urilor TR stricte aperiodice, invocate de mecanismul de întreruperi, afectează puternic predictibilitatea şi funcționarea deterministă a sistemului (vezi discuția din Capitolul 3), făcând practic imposibilă implementarea unui algoritm de planificare fezabil şi complet predictiv pentru STR stricte (după cum se va vedea şi în Capitolul 5).

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

- Chiar dacă o anume aplicație 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 specificațiile temporale de execuție 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 soluție fezabilă şi relativ uşor de implementat, aşa cum se va vedea în continuare.
- Două din dezavantajele majore ale acestui tip de abordare constau în scăderea drastică a eficienței 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 operării sistemului pe care o implică 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 execuție 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 execuție la începutul fiecărui ciclu de execuție (la începutul fiecărei perioade): $T_{dy,k}^{M_i} = 0, \forall\, k \in \{1, \ldots, N^{M_i}\}$.

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 relația[1]:

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

Demonstrație

Vom demonstra teorema luând în considerare cazul cel mai defavorabil care poate să apară relativ la momentul începerii execuției lui M_i față de momentul apariției lui S_j: semnalul apare în instanța imediat următoare începerii execuției din ciclul curent a lui M_i, iar intervalul de timp dintre două execuții consecutive este maxim (vezi Figura 4-5).

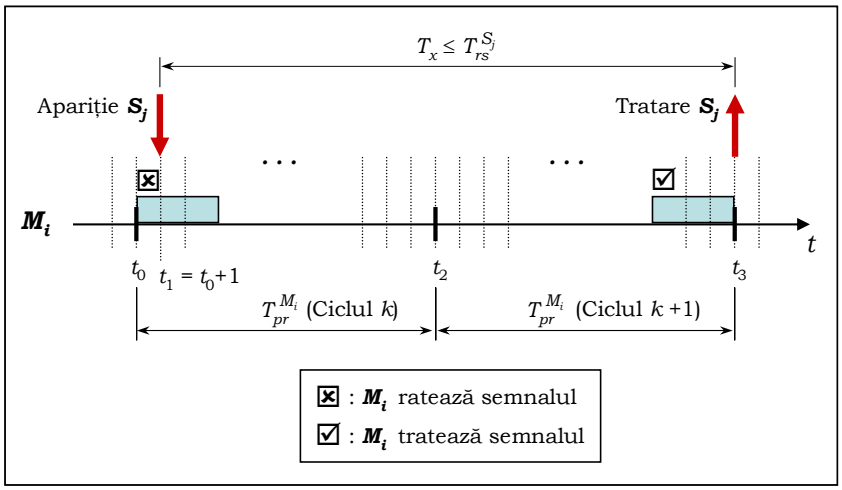

Figura 4-5. Cazul cel mai defavorabil tratării semnalului S_j cu un task M_i

Din figura de mai sus se observă că execuția k a lui M_i ratează apariția lui S_j, starea acestuia (informația furnizată de semnal) fiind reținută într-o structură tampon a sistemului, așa cum am descris mai sus, pentru tratarea în cadrul unei execuții ulterioare a lui M_i. În situația cea mai defavorabilă, execuția (k + 1) a lui M_i are loc la sfârşitul ciclului de execuție următor (la sfârşitul perioadei următoare), pe când execuția curentă are loc la începutul perioadei.

[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.

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

$$T_x = 2 \cdot T_{pr}^{M_i} - 1 \qquad (4\text{-}17)$$

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), obținem relația (4-16), cerută a fi demonstrată în enunțul 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 condițiile în care se specifică doar parametrii temporali ai semnalelor. Din punctul de vedere al programatorului de aplicații, 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 4-5.

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 execuție a ambelor task-uri o considerăm $T_{ex}^{M_1} = T_{ex}^{M_2} = 3$ unități de timp.

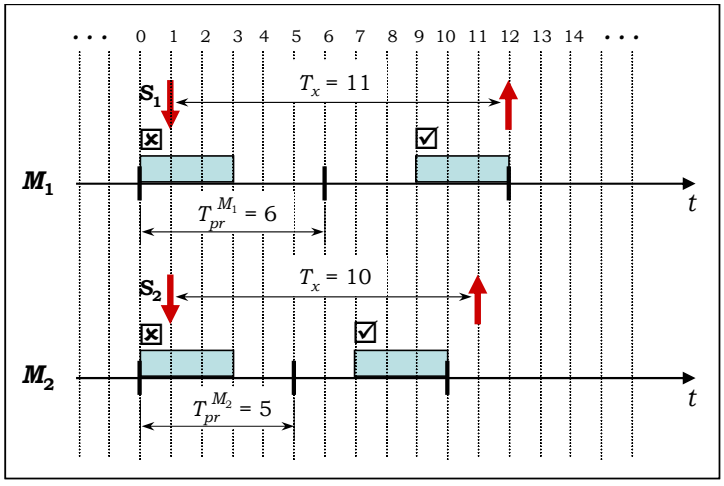

Figura 4-6. Task-urile M_1 şi M_2 ce tratează semnalele S_1, respectiv S_2

Utilizând Teorema 4-1 pentru determinarea parametrilor temporali ai lui M_1 şi M_2 (a perioadelor acestora), rezultă (vezi Figura 4-6):

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

pentru M_1, şi:

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

❑

Teorema 4-1 permite tratarea cu ajutorul task-urilor TR stricte periodice a semnalelor aperiodice de intrare pentru care se specifică intervalul de răspuns al sistemului (cazul cel mai frecvent întâlnit în practică: pentru utilizator/programatorul de aplicații este mai uşoară şi mai intuitivă specificarea intervalului de răspuns al unui RTS la un semnal aperiodic de intrare, decât determinarea intervalului minim de timp între oricare două apariții consecutive ale acestuia – perioada semnalului).

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 apariții 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 apariție 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 4-5 şi Figura 4-6, cu $T_x = T_{pr}^{S_j}$.

Dacă sunt specificați 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 relația

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

care afirmă că, practic, intervalul alocat tratării lui S_j de către STR nu poate depăşi ca durată intervalul celei mai apropiate apariții 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 apariții 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 execuției lui M_i, S_j este tratat în cel de-al doilea ciclu de execuție, înainte unei noi apariții. Problema acestei abordări însă constă în scăderea drastică a eficienței sistemului – prețul plătit de sistemele TR stricte în schimbul garantării îndeplinirii tuturor specificațiilor temporale ale aplicației, în orice condiții de operare.

Exemplul 4-6.

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 execuție a lui M_1 o considerăm constantă pentru toate ciclurile de execuție: $T_{ex}^{M_1} = 2$.

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

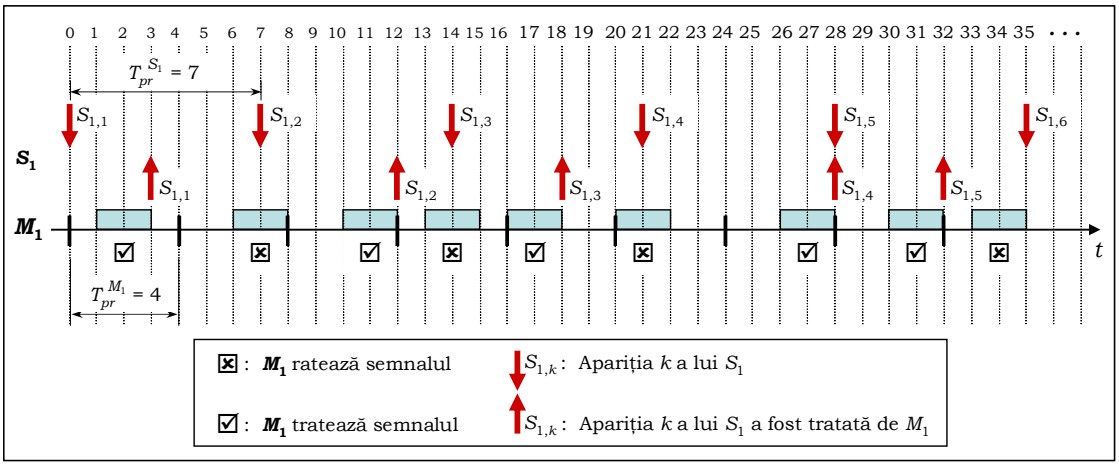

Figura 4-7. Exemplificare a tratării unui semnal de intrare periodic

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

Total apariții S_1	6
Total execuții M_1	9
Execuții M_1 care tratează S_1	5
Execuții M_1 care ratează S_1	4
Eficiență execuție M_1	55 %

❑

Pentru creşterea eficienței tratării semnalelor de intrare periodice, se pot utiliza task-uri TR periodice, cu aceeaşi perioadă ca a semnalului de intrare, cu condiția sincronizării apariției semnalului cu execuția 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 aplicații 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 apariția semnalului şi execuția 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 apariție 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. În Exemplul 4-2, se observă că atât operarea subsistemului de conversie A/N, cât şi activarea semnalului CDOUT, sunt comandate (sincronizate) de către STR, prin activarea liniei STC, cu o perioadă care trebuie să fie mai mare sau egală cu durata unui ciclu de conversie al circuitului DAC.

Observație

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 execuții de arbitrarea şi tratarea, una câte una, a întreruperilor ce apar în sistem.

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

- durata de execuție a serverului de întreruperi va trebui să fie cel puțin egală cu cea mai mare durată de execuție corespunzătoare tratării oricărui semnal aperiodic din cele gestionate de server,
- perioada serverului va trebui să fie suficient de mică pentru a putea satisface specificațiile 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 specificații temporale mai diferite între ele, cu atât mai dificilă este planificarea şi execuția 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 față, 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 specificații complete şi stricte de comportare în timp şi cu planificare şi execuție în context non-preemptiv, destinat a servi ca element de bază în specificarea, proiectarea şi dezvoltarea sistemelor şi aplicațiilor TR stricte, pentru care determinismul şi predictibilitatea operării sunt cerințe esențiale.

Din perspectiva arhitecturii aplicațiilor TR stricte, cât şi din punct de vedere funcțional, 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 instrucțiuni care prelucrează intrările, variabilele definite local şi variabilele globale, şi un set de ieşiri (parametri de ieşire). Din punct de vedere al specificațiilor şi al comportamentului în domeniul timp, parametrii de intrare/ieşire ai ModX-ului sunt semnale interne ale sistemului. În plus, un ModX poate interacționa 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 aplicații TR reprezentate sub forma unui graf de program, direcționat, 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 mulțimea nodurilor grafului $\boldsymbol{G}$ (ModX-urile aplicației) şi $\boldsymbol{V} = \{(M_i, M_j) \mid M_i, M_j \in \boldsymbol{M}\ ;\ 1 \le i, j \le |\boldsymbol{M}|\}$ este mulțimea arcelor lui $\boldsymbol{G}$, constând din perechi ordonate de noduri.

Definiția 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 specificațiilor 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 instrucțiuni care procesează informația în cadrul M_i, reprezentând specificarea funcțională 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 obține un sistem TR care să opereze în condiții maxime de predictibilitate şi determinism, şi ale cărui task-uri să poată fi executate cu respectarea în orice condiții 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] (vezi discuțiile din Capitolul 3 şi 6).

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 aplicației, cât şi analiza fezabilității planificării task-urilor înaintea încărcării şi execuției lor efective în sisteme TR a căror fiabilitate de operare este critică. Un alt avantaj constă în eliminarea problemelor de tip inversare a priorității, tipice sistemelor TR care implementează tehnici de planificare şi execuție preemptive, bazate pe priorități.

Abordarea modulară

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

- Creşterea flexibilității în procesul de dezvoltare a aplicațiilor TR. Dezvoltarea modulară şi vizuală a aplicațiilor, utilizând reprezentări de tip graf, uşurează munca de proiectare, specificare, validare şi analiză a acestora.
- Evidențierea rapidă a dependențelor de program (control şi date) ale task-urilor aplicației.
- Rezolvarea simplă a comunicației şi sincronizării între task-urile aplicației. Există două mecanisme puse la dispoziția programatorului de aplicații: transmiterea de parametri de intrare/ieşire şi utilizarea (limitată) a variabilelor globale.
- Ținând cont de faptul că task-urile aplicației sunt periodice, este mai uşoară şi mai intuitivă vizualizarea execuției acestora ca module în cadrul aplicației.

Aspectul funcțional şi evaluarea duratei de execuție

Din punct de vedere funcțional, ModX-urile reprezintă secvențe de cod ce se execută fără blocare şi în regim non-preemptiv. Prin urmare, operațiile implementate de un ModX sunt *operații 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 execuție a task-urilor. Parametrul temporal $T_{ex,k}^{M_i}$ specifică durata de execuție a task-ului M_i, care variază de la un ciclu k de execuție la altul (vezi

Definiția 4-2 şi Tabelul 4-3). Pentru a se putea dezvolta algoritmi fezabili de planificare, este necesară utlizarea pentru fiecare task a unei durate de execuție constante, $T_{ex}^{M_i}$, care să nu varieze de la un ciclu de execuție la altul. Pe de altă parte, planificarea task-urilor unui STR strict trebuie să fie fezabilă pentru cele mai dezavantajoase situații de operare ale sistemului şi nu doar pentru cazurile de încărcare medie, de exemplu. Din aceste considerente, durata de execuție a unui task TR strict, pentru fiecare ciclu de execuție, este considerată ca fiind egală cu *WCET* (*Worst Case Execution Time*), adică durata de execuție cea mai defavorabilă [Puschner 89, Puschner 91, Park 91, Burns 91, Chapman 94, Parashkevov 94]. Durata *WCET* pentru un task oarecare se determină ca fiind suma timpilor de execuție ai instrucțiunilor ce compun calea cea mai lungă de execuție a task-ului şi considerând condițiile cele mai defavorabile de operare.

WCET nu poate fi determinat fără o serie de restricții impuse limbajului de programare cu ajutorul căruia se specifică comportamentul funcțional 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 restricții, 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].

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

- Utilizarea *recursivității* – este interzisă, din cauza cerințelor de spațiu impredictibile pe care le produce (dimensiunea memoriei necesare).
- *Buclele de program* trebuie să fie limitate ca durată totală şi ca spațiu. Cu alte cuvinte, trebuie ca numărul total de iterații ale fiecărei bucle să fie bine determinat. De asemenea, buclele încuibate (*nested loops*) se vor supune acelorlaşi restricții, inclusiv unei limitări a nivelurilor de încuibare, pentru o analiză mai uşoară a programului.
- Utilizarea *instrucțiunilor de ramificare* de tip `GOTO` sunt permise doar în condițiile în care au ca efect salturi de tip "înainte" ale execuției. În acest fel se păstrează reductibilitatea grafului dependențelor 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 funcții* 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, funcțiile 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 execuție, 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 țintă şi să respecte restricțiile enumerate mai sus. Astfel, într-o primă fază se utilizează direct limbajul de asamblare, şi intenționăm adaptarea unei versiuni de "ANSI C" pentru programarea aplicațiilor TR. Compilarea/asamblarea fiecărui ModX în parte se face cu opțiuni de generare de cod relocabil (bibliotecă de module).

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

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

Comunicația între ModX-urile aplicației 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 execuție până când toți parametrii săi de intrare nu au fost actualizați de ModX-urile care au aceiaşi parametri ca parametri de ieşire.

$\boldsymbol{P_{GLB}}$ reprezintă setul acelor variabile globale ale aplicației care sunt accesate de către un ModX. Variabilele globale reprezintă un al doilea mecanism de sincronizare şi comunicare inter-task pus la dispoziția programatorului de aplicație. Datorită faptului că ModX-urile se execută în context non-preemptiv şi, astfel, implementează operații atomice, nu sunt necesare mecanisme suplimentare de control şi sincronizare a acceselor concurente la variabilele globale ale aplicației. Se recomandă, totuşi, utilizarea cu atenție a variabilelor globale pentru că ele practic ascund dependența normală de date dintre task-urile aplicației, cu implicații în construirea unui graf al dependențelor de program realist. O situație în care se recomandă utilizarea variabilelor globale este aceea în care aplicația 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 execuție la alta a ModX-urilor, sub formă de parametri de intrare/ieşire.

Observație

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 execuție a sa, la alta. Această operație 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 inițializare a aplicației (incluzând setarea valorilor inițiale pentru astfel de parametri), care să se execute primul, înaintea celorlaltor ModX-uri. Acesta are un regim special de execuție, nefiind periodic ci executându-se o singură dată.

Tratarea şi generarea semnalelor

ModX-urile interacționează cu semnalele de intrare/ieşire ale STR în maniera descrisă în secțiunea precedentă. Din punct de vedere al programatorului de aplicații,

setul de semnale $\boldsymbol{S} = \{\boldsymbol{S_{IN}}, \boldsymbol{S_{OUT}}\}$ cu care interacționează un ModX M_i este perceput la nivelul specificării comportamentului temporal al semnalelor. Informația 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:

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

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

Specificarea comportamentului temporal al ModX-urilor

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

- $T_{pr}^{M_i}$ este perioada lui M_i – caracteristică esențială şi obligatorie pentru toate ModX-urile, fiind task-uri periodice. Perioada este constantă de-a lungul operării aplicației, deci nu depinde de ciclul curent k de execuție a lui M_i ;
- $T_{ex}^{M_i}$ este durata de execuție 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 execuția lui M_i în cadrul perioadei acestuia. $T_{dl}^{M_i}$ este un parametru opțional, introdus pentru cazurile în care este necesar ca execuția lui M_i să se finalizeze cu un interval de timp oarecare înainte de terminarea perioadei, pentru toate ciclurile de execuție;
- $T_{dy}^{M_i}$ este intervalul de întârziere a planificării şi execuției lui M_i în cadrul perioadei. $T_{dy}^{M_i}$ este, la rândul lui un parametru opțional, ce poate fi utilizat în cazurile în care se doreşte ca planificarea şi execuția 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 execuții ale lui M_i. Este un parametru opțional, care tratează trei cazuri:
  - M_i are un număr nedefinit de execuții (execuție continuă): $N^{M_i} = \infty$,

- M_i are un număr finit, bine precizat de execuții: $N^{M_i} < \infty$. În acest caz, M_i interacționează cu semnale (generează sau tratează semnale) cu apariție 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 execuție de către executivul/dispatcher-ul sistemului.

Observație

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

i.) În timpul execuției 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 execuțiile lui Mi pe parcursul operării. Când N^{M_i} ajunge la 0, executivul nu-l va mai lansa în execuție pe M_i.

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

Definiția 4-10.

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

❑

Figura 4-8 exemplifică operarea pentru două perioade consecutive (două cicluri de execuție) a unui ModX oarecare M_i ce are specificați toți parametrii săi temporali (*ModX generic*).

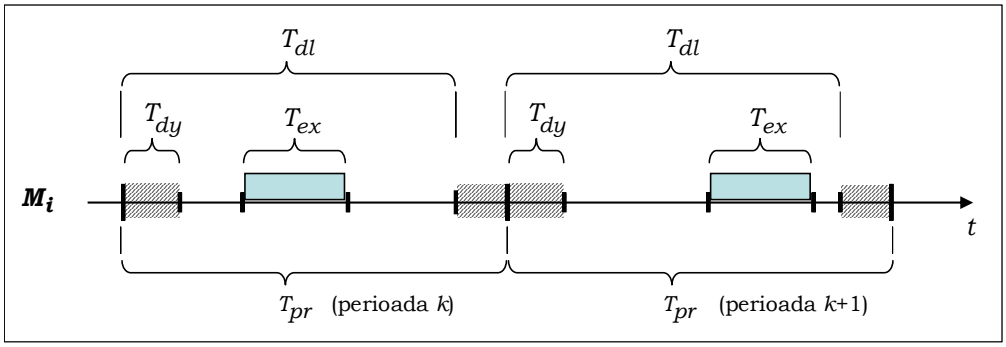

Figura 4-8. ModX-ul generic M_i şi parametrii sări temporali

Specificarea temporală minimală a unui ModX M_i, constă din cei doi parametri de bază: perioada $T_{pr}^{M_i}$ şi durata de execuție $T_{ex}^{M_i}$. Pe baza acestor parametri, algoritmii de planificare (non-preemptivă) pot analiza fezabilitatea planificării setului de ModX-uri al sistemului TR strict şi, în caz că există o variantă de planificare fezabilă, o pot obține. Setarea celorlalți parametri temporali se va face implicit, în faza de verificare/validare a aplicației, după cum urmează:

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

Relația (4-23) specifică pentru M_i, un interval limită egal cu perioada, un interval de întârziere nul şi un număr nedefinit de execuții (operare continuă). Figura 4-9 exemplifică operarea pentru două perioade consecutive a unui ModX M_i ce are specificați doar cei doi parametri temporali de bază (*ModX simplu*).

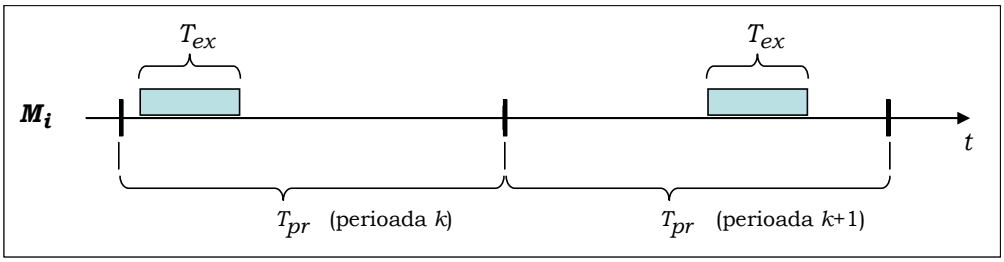

Figura 4-9. ModX-ul simplu M_i şi parametrii săi temporali

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

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

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

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

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 relațiilor (4-14), (4-15), respectiv (4-16) şi (4-18), adaptate definirii ModX-urilor ca task-uri TR stricte, periodice şi cu parametrii temporali (mulțimea $\boldsymbol{T}$ din (4-21)) constanți pentru toate ciclurile de execuție.

5 Planificarea aplicațiilor TR stricte

5.1 Introducere

Una dintre problemele centrale de interes 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 implementarea 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 aplicațiilor TR de achiziție şi prelucrare numerică a datelor în ritm continuu, neîntrerupt (de exemplu, în aplicațiile 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 menționat mai sus – generarea de probleme computaționale de ordin strict nepolinomial. Alte dezavantaje includ:

- Necesitatea găsirii unei partajări optime a secvenței de cod a programului în task-uri (problema *granularității* task-urilor), în aşa fel încât planificarea non-preemptivă a acestora să poată fi efectuată în condiții optime de eficiență. 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 aplicației;
- Lipsa de flexibilitate a abordării non-preemptive. Odată cu eliminarea întreruperilor din sistem, se elimină şi capacitatea sistemului de a face față unor configurații noi, asincrone, ale mediului în care operează şi cu care interacționează. Din aceste motive, vor trebui prevăzute şi tratate apriori, încă din faza de specificare şi proiectare, toate posibilitățile şi configurațiile de operare ale sistemului;
- Creşterea ineficienței 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 interacționează cu semnale (evenimente) asincrone;
- Dificultatatea găsirii unor condiții de testare rapidă a planificabilității şi de găsire a planificării optime a unui set de task-uri. Pentru multe modele ale aplicațiilor, testarea planificabilității setului aferent de task-uri şi găsirea unui algoritm optim de planificare reprezintă probleme computaționale 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 condiții maxime de predictibilitate şi determinism, cu garantarea respectării termenelor de timp impuse task-urilor TR stricte, spre deosebire de algoritmii de planificare ce permit existența întreruperilor, a evenimentelor şi task-urilor asincrone, sporadice.

Exemplul 5-1.

Considerăm o stație terestră de emisie-recepție care comunică permanent cu un satelit nestaționar aflat pe o orbită în jurul Pământului. Două dintre funcțiunile principale executate de sistemul digital de control şi procesare al stației sunt comunicația 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 funcții se desfăşoară în regim permanent, dar, dacă partea de comunicație poate fi modelată în mod direct ca un task periodic (set de task-uri periodice), funcția de orientare a antenei poate fi modelată ca un task asincron: necesitatea reorientării antenei fiind semnalată de scăderea sub un anumit prag a intensității semnalului recepționat de la satelit. Evident, sistemul de control şi procesare al stației este un sistem TR strict.

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

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

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

❑

În capitolul de față 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 inițiale, 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 execuție constantă de la un ciclu de execuție la altul;
iii.) Înainte de execuția efectivă a sistemului de task-uri în cadrul STR, are loc o analiză a fezabilității planificării acestora. Numim această operație *analiză offline*;
iv.) STR operează într-un mediu determinist şi toate specificațiile 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.) Toți algoritmii de planificare studiați vor implementa o *planificare fără inserție de timp liber*, adică nu vor introduce în planificare intervale de timp liber în situațiile 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.

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

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

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 execuție, $T_{dl}^{M_i}$ este intervalul limită, $T_{dy}^{M_i}$ este intervalul de întârziere şi N^{M_i} este numărul total de execuții.
- Între parametrii lui M_i se stabilesc următoarele relații:

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

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

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

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

- Momentul primei invocări a fiecărui ModX M_i este:

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

- Execuția lui M_i nu este condiționată de execuțiile altor ModX-uri, adică nu există dependețe de control sau de date între oricare două ModX-uri din $\boldsymbol{M}$.

❑

Setul $\boldsymbol{T}$ al parametrilor temporali din (5-1) specifică un ModX generic, pentru care, în mod obligatoriu, durata de execuție este nenulă şi mai mică sau egală cu perioada. Relațiile (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 execuție este nelimitat.

Definiția 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 specificațiile de comportare temporală pe toată durata operării sistemului.

❑

Există două clase de algoritmi de planificare ce pot fi aplicați 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 instanță 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 priorităților în cadrul unui set de ModX-uri este asignarea unei priorități invers proporționale 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 cerința 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, considerați a fi cei mai eficienți pentru tratarea execuțiilor 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 Definiția 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.

Demonstrație

Lema susține 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*** conține ModX-uri simple, cu caracteristicile date de Definiția 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 execuție: $T_{pr}^{M_i}$, 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 configurație 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 fezabilității 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 selecție 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.

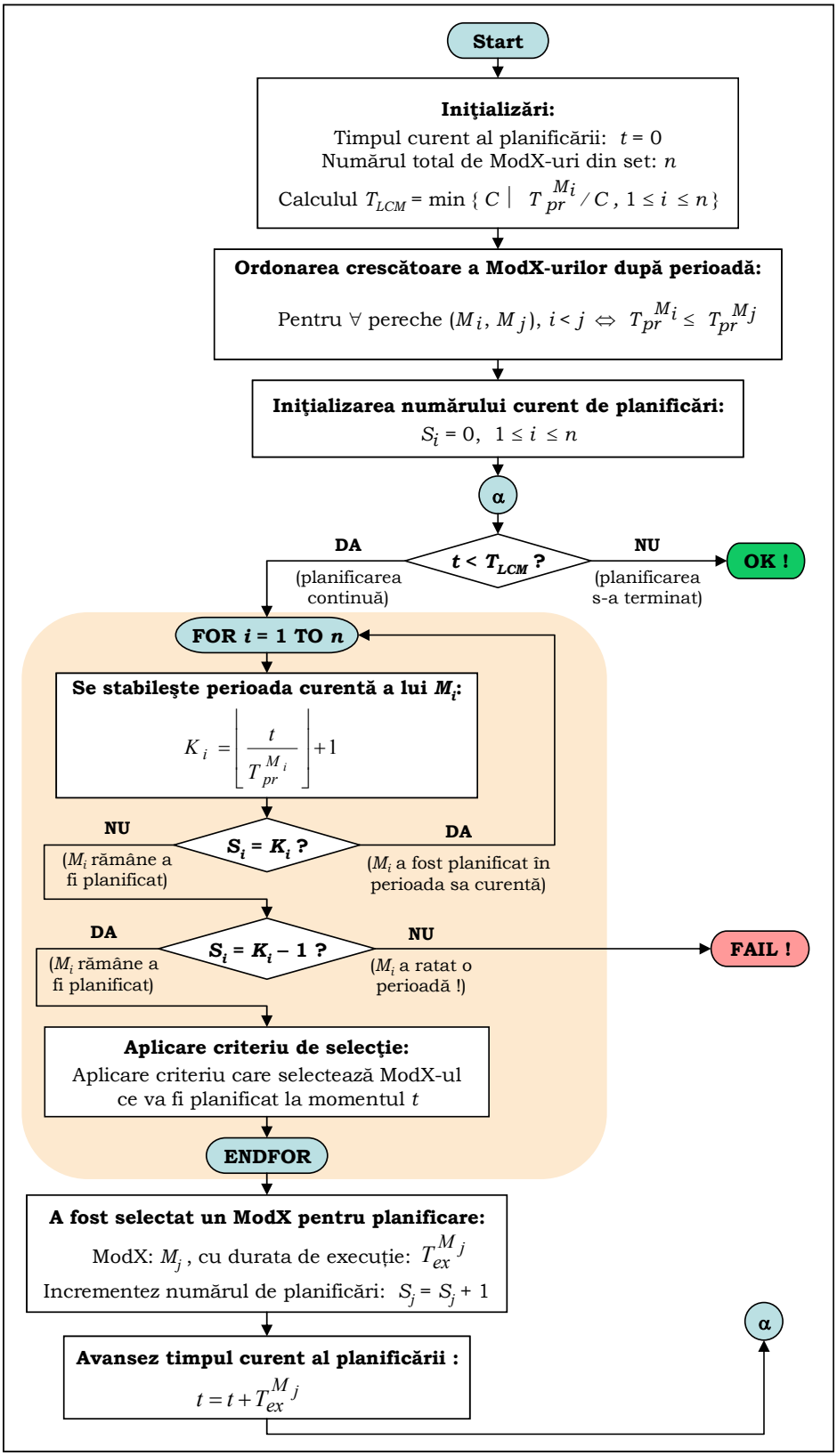

Figura 5-1. Organigrama generală a algoritmilor de planificare non-preemptivă a unui set de ModX-uri independente

5.2.2 Algoritmul de planificare MLFNP

Pentru selecția 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):

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

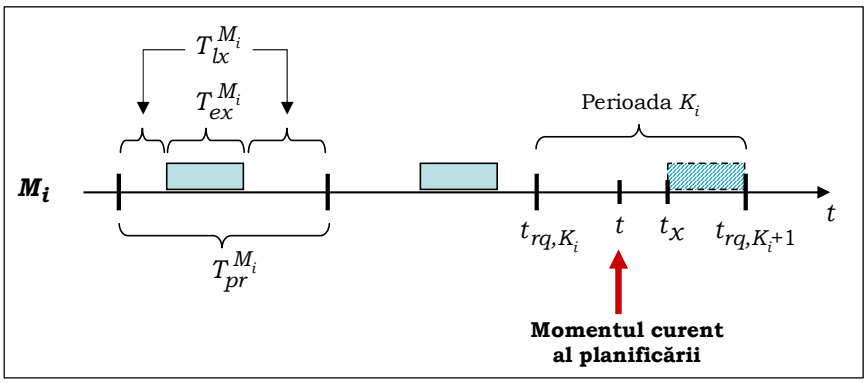

Figura 5-2. Intervalul disponibil planificării în cazul ModX-urilor simple

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

Criteriul de selecție 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 dispoziție 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 "urgența") planificării lui M_i este mai mică în comparație 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 execuție, 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 selecție 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 dispoziție 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 funcție 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 funcții, câte una pentru fiecare ModX din $\boldsymbol{M}$ (unde $n = |\boldsymbol{M}|$). În finalul buclei de selecție, 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 selecția 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 execuție 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 5-1.

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 planificabilității 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 inițializarea parametrului S_i (numărul total de planificări pentru ModX-ul M_i, până la momentul curent):

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

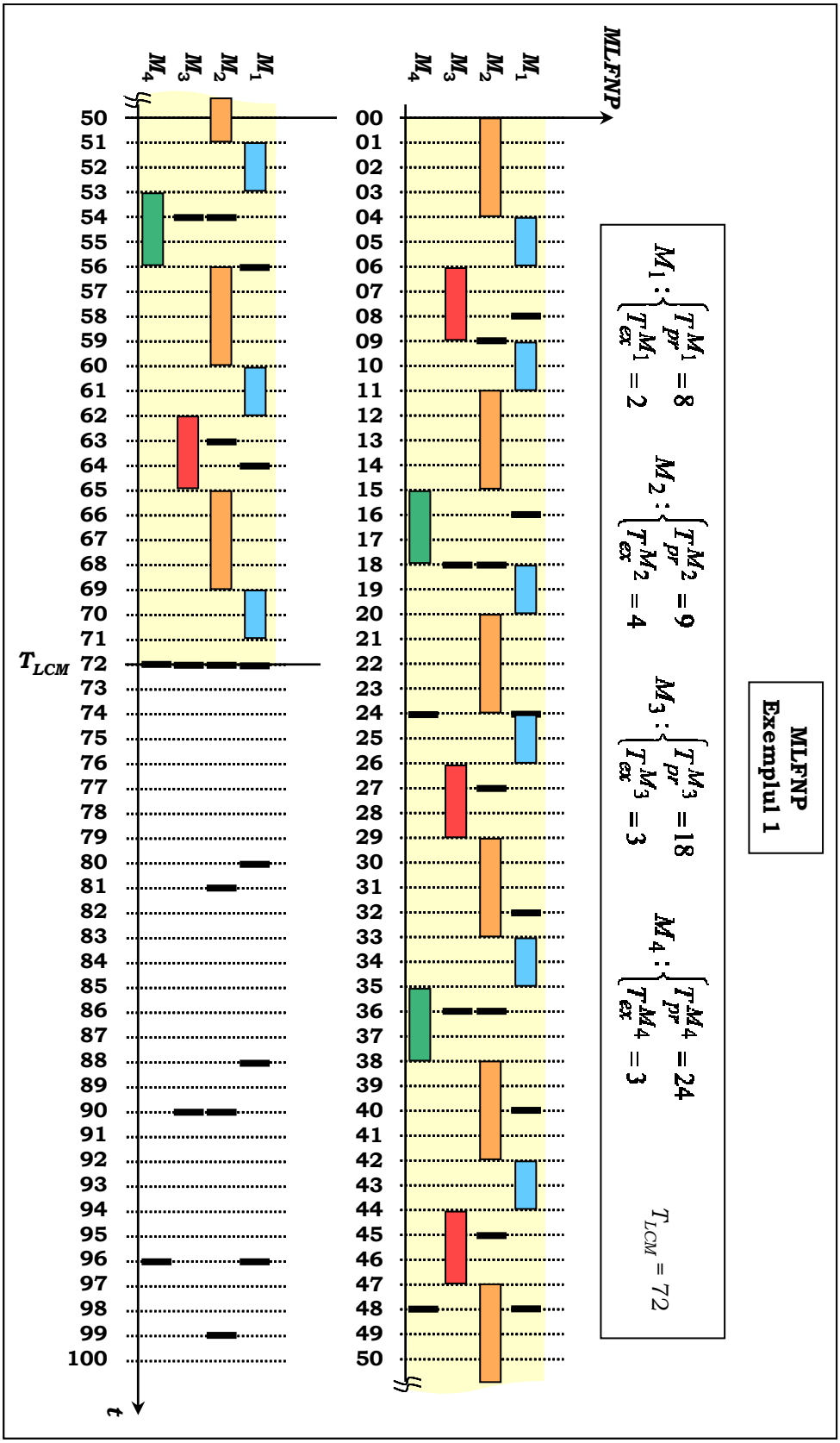

Figura 5-3. Exemplu de planificare *MLFNP* a unui set de ModX-uri

Primul pas îl reprezintă stabilirea pentru fiecare ModX a numărului ciclului de execuție 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 relației (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 5-2.

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 planificabilității 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:

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

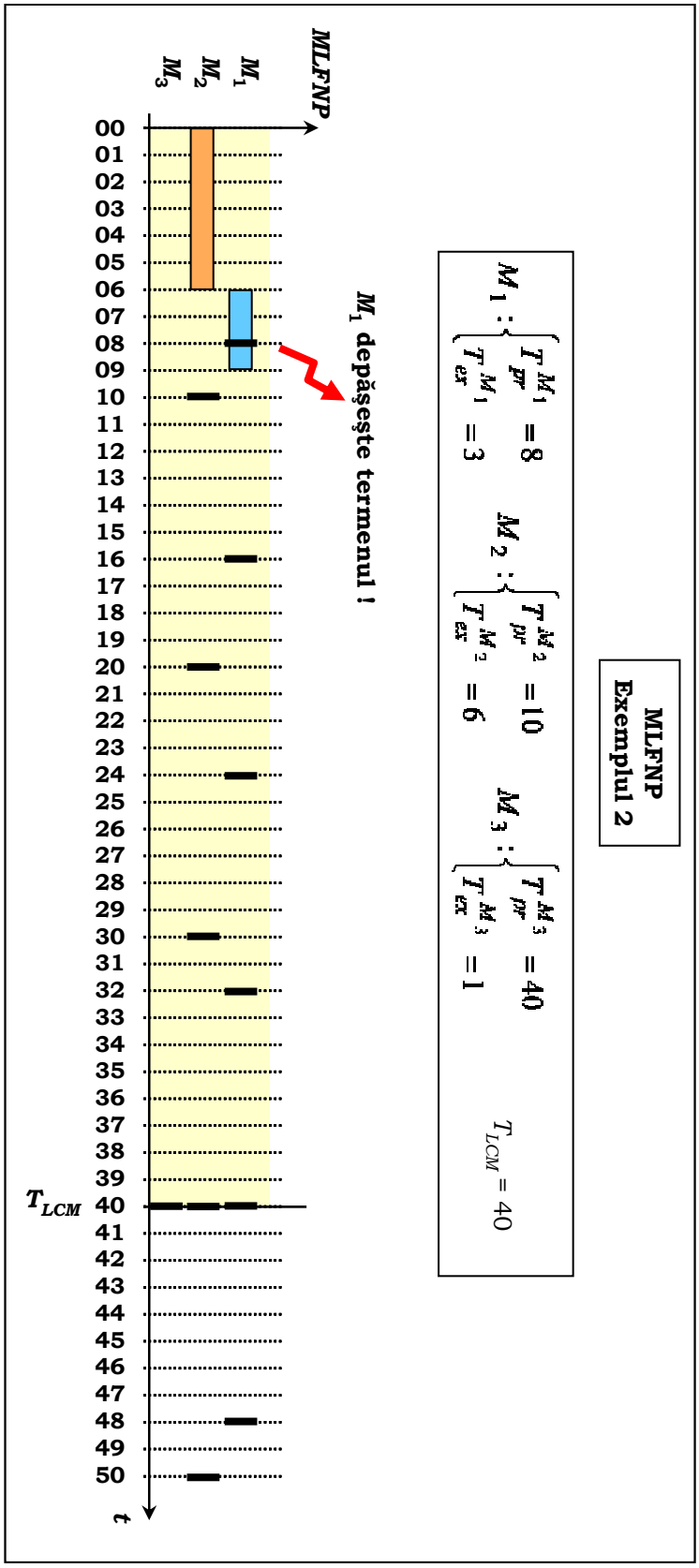

Figura 5-4. Exemplu de planificare MLFNP a unui set de ModX-uri

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 5-2 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 secțiunea următoare). Vom spune că setul de ModX-uri din Exemplul 5-2 *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 puțin 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 5-1 şi 5-2, pentru a ilustra inclusiv diferențele de abordare a planificării (prin criteriile de selecție a ModX-ului care urmează a fi planificat la momentul curent).

Exemplul 5-3.

Considerăm un acelaşi set ***M*** = $\{M_1, M_2, M_3, M_4\}$ de ModX-uri simple, independente, ca în Exemplul 5-1:

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

Figura 5-5 ilustrează analiza offline a planificabilității 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 selecție a ModX-urilor care nu au fost încă planificate.

❑

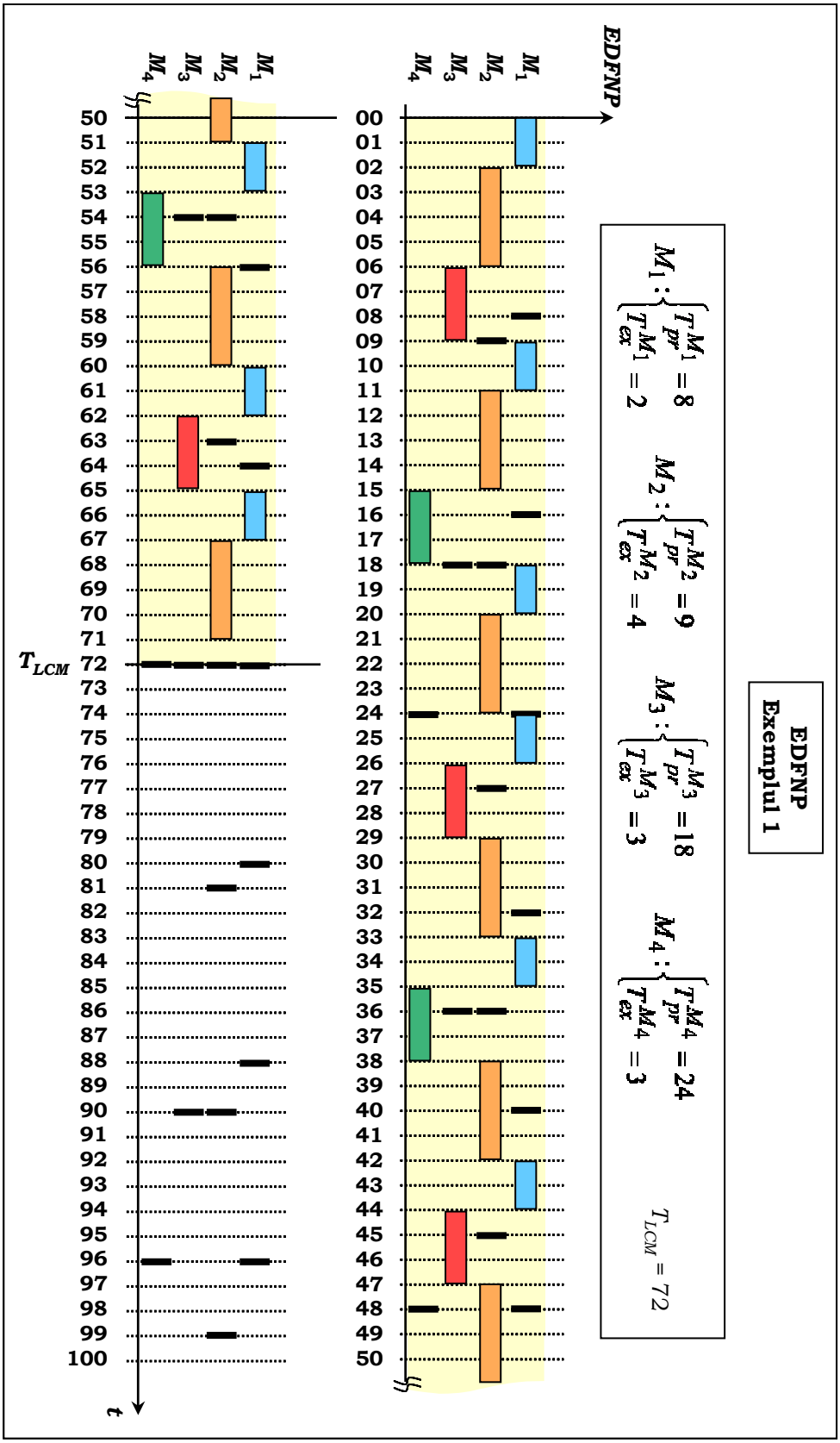

Figura 5-5. Exemplu de planificare *EDFNP* a unui set de ModX-uri

Exemplul 5-4.

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

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

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

❑

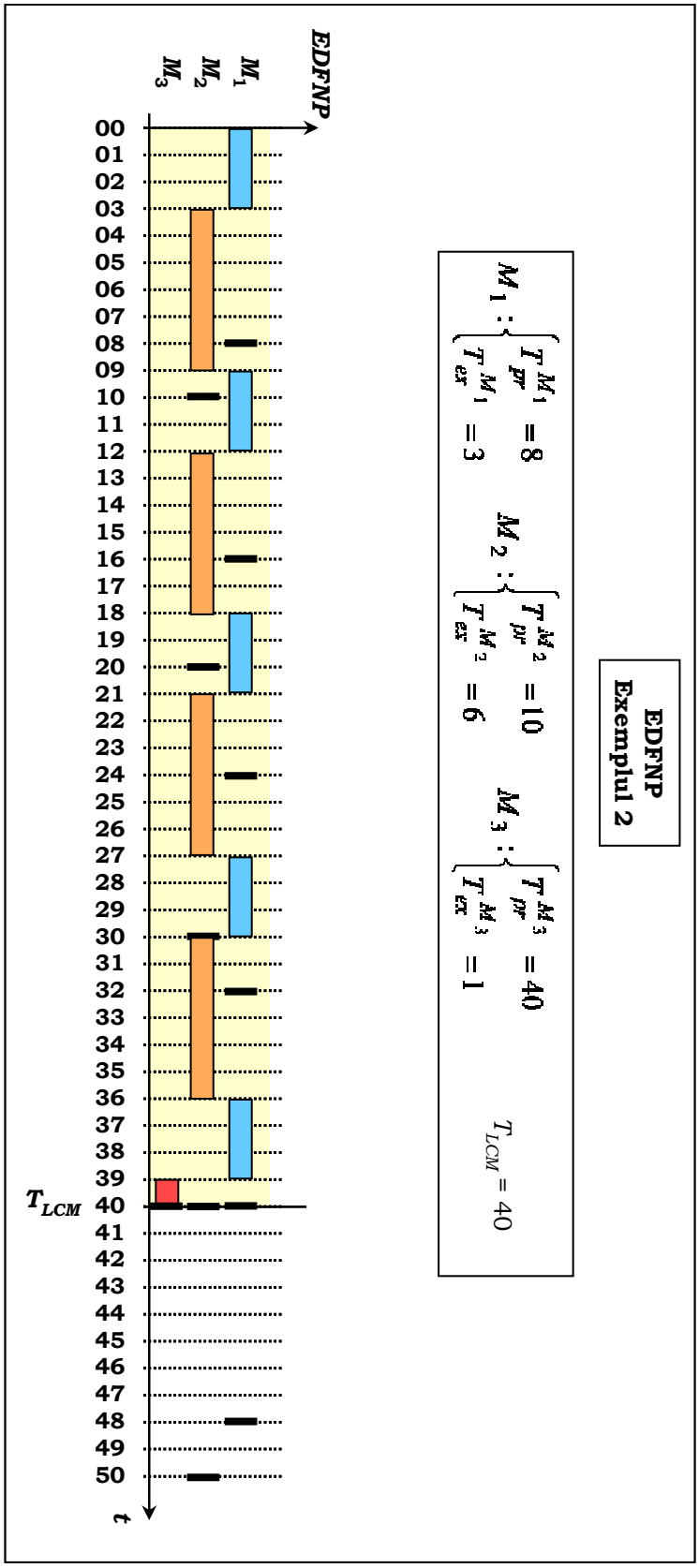

Figura 5-6. Exemplu de planificare *EDFNP* a unui set de ModX-uri

5.2.4 Condițiile de planificabilitate

În cadrul analizei offline a unei aplicații TR, determinarea fezabilității 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 condiții – *condițiile de planificabilitate*. Evaluarea condițiilor de planificabilitate înaintea aplicării algoritmului propriu-zis prezintă avantajul unei decizii rapide, obținute cu eforturi relativ mici (resurse de calcul, timp, etc.).

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

- *Condiții de necesitate*: dacă setul de task-uri nu le îndeplineşte, atunci în mod sigur acesta nu este fezabil pentru planificare;
- *Condiții de suficiență*: 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, condițiile de suficiență sunt foarte dificil de determinat – dacă ele există [Kim 80, Jeffay 91]. Pot fi stabilite însă o serie de condiții de necesitate, care să fie utile în identificarea rapidă a unui set de task-uri ce nu este fezabil. O primă condiție 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 enunțată şi demonstrată în [Jeffay 91]. Vom prezenta în continuare cele două condiții de necesitate a planificabilității pentru cazul general al seturilor de task-uri ce vor fi planificate în context non-preemptiv şi le vom analiza sub aspectul valabilității 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 secțiunea de față: amândouă iau în considerare task-uri periodice, caracterizate din punct de vedere temporal prin doi parametri – perioada şi durata de execuție. Diferența constă din faptul că în [Jeffay 91] se tratează seturile de task-uri la modul general, făcându-se distincție 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 mulțimea 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, condițiile de necesitate pentru planificarea non-preemptivă sunt enunțate în [Jeffay 91] după cum urmează:

"Fie $\boldsymbol{M} = \{M_1, M_2, \ldots, M_n\}$ 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 $T_{pr}^{M_i} \leq T_{pr}^{M_j}$).

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

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

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

$$L \ge T_{ex}^{M_i} + \sum_{j=1}^{i-1} \left\lfloor \frac{L-1}{T_{pr}^{M_j}} \right\rfloor \cdot T_{ex}^{M_j} \qquad (5\text{-}14)$$

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

CN1 este o condiție de bază a oricărei planificări de task-uri pe sisteme cu un singur procesor şi exprimă cerința ca suma fracțiunilor de timp procesor consumate de fiecare task din setul ***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 secțiunea de față. Cu ajutorul acestei condiții, 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 5-1 şi 5-3, respectiv din Exemplul 5-2 şi 5-4 îndeplinesc **CN1**, având utilizarea procesor egală cu 0.986, respectiv 1.

Condiția **CN2** reflectă restricția impusă de planificarea non-preemptivă şi fără inserarea de timp liber pentru seturile de task-uri. Astfel, dacă un set ***M*** de task-uri este fezabil, atunci orice set concret de task-uri ***W*** generat din ***M*** poate fi planificat cu un anume algoritm non-preemptiv, şi în consecință, îndeplineşte relația (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** enunțată în [Jeffay 91] nu este aplicabilă, aşa cum o arată şi exemplul următor.

Exemplul 5-5.

Considerăm un set ***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 planificabilității setului ***M*** utilizând algoritmul *EDFNP* pentru intervalul [0, T_{LCM}), unde T_{LCM} = 90, cu rezultate pozitive. În consecință setul ***M*** este fezabil.

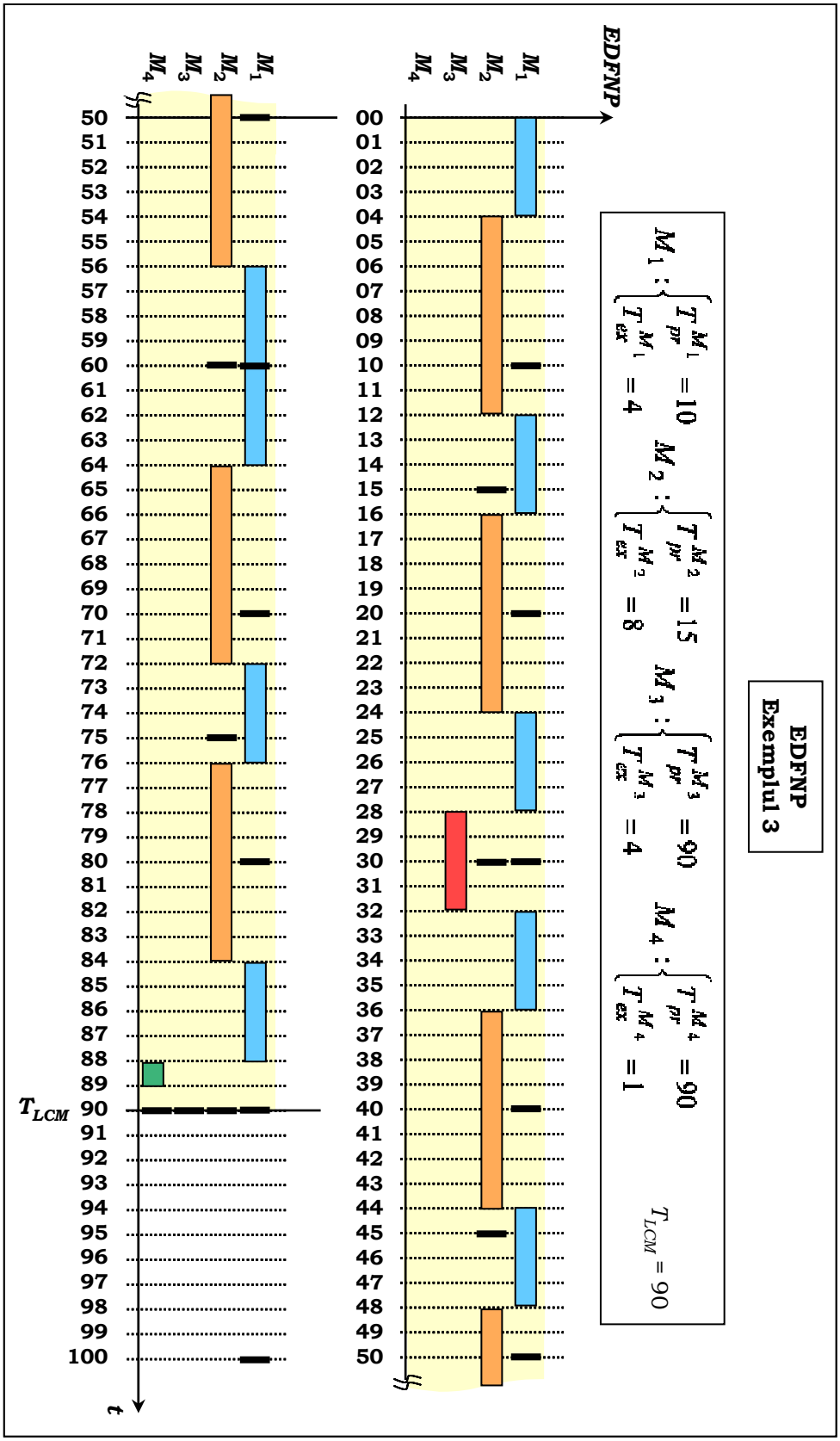

Figura 5-7. Planificarea EDFNP a setului M

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

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

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

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

În consecință, condiția **CN2** nu este îndeplinită, deşi setul ***M*** este totuşi fezabil.

❑

5.3 Planificarea non-preemptivă a ModX-urilor independente, în cicluri cu număr constant de execuții

În secțiunea precedentă am tratat analiza offline a planificabilității 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 aplicației 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 5-4, lista generată la analiza offline a planificabilității e prezentată în Tabelul 5-1 şi are un total de 10 elemente, acoperind ca timp al planificării intervalul $[0, T_{LCM}) = [0, 40)$.

Din punct de vedere practic, al implementării planificării pe un STR real, se pune problema resurselor necesare, în speță, a spațiului de memorie necesar stocării în sistem a listei cu planificarea ModX-urilor. Dimensiunile listei de planificare variază puternic în funcție de setul de ModX-uri care se doreşte a fi planificat şi de caracteristicile temporale ale acestora. De exemplu, în cazul în care setul conține relativ multe ModX-uri şi cu perioade puternic diferite între ele (fără divizori comuni), iar duratele de execuție 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, spațiul de memorie necesar planificării acestui set poate depăşi cu uşurință resursele disponibile ale sistemului.

Tabelul 5-1. Lista planificării EDFNP pentru Exemplul 5-4

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 față presupune existența 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 conține ModX-urile planificate împreună cu momentele de start ale execuției acestora, şi pe care o denumim *tabela de dispatch*, este de fapt un buffer circular, 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 execuție a câte unui ModX din tabelă, la momentul specificat de planificare (vezi). Notăm cu λ dimensiunea utilă a *tabelei de dispatch*, adică numărul elemente ale tabelei ce conțin ModX-uri ale aplicației, planificate de către MS în cadrul unui *ciclu de planificare*.

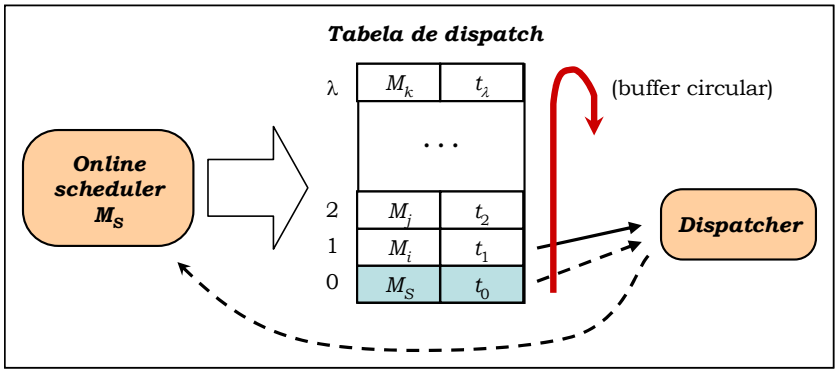

Figura 5-8. Structura de principiu a *tabelei de dispatch*

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

- Execuția lui M_S se desfăşoară tot context non-preemptiv;
- $T_{ex}^{M_S}$ este durata de execuție;

- Perioada lui M_S, $T_{pr}^{M_S}$, nu se defineşte. Aşadar, execuția M_S nu este periodică în timp, în schimb *online scheduler-ul* prezintă o execuție ce se repetă ciclic de fiecare dată ce au fost planificate (executate) λ ModX-uri ale aplicației;
- Algoritmul de planificare implementat de către *scheduler-ul online* M_S va fi o versiune similară a algoritmului utilizat în analiza offline a planificării aplicației, pentru a avea garanția 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 *MEDFNP* (*Modified Earliest Deadline First Non-Preemptive*), cu organigrama operării de principiu prezentată în Figura 5-9. Algoritmul *MEDFNP* planifică prima dată (la momentul $t = 0$) *online scheduler-ul*, avansează timpul de planificare cu $t = t + T_{ex}^{M_S}$, după care aplică *EDFNP* pentru următoarele λ planificări de ModX-uri ale aplicației (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}.

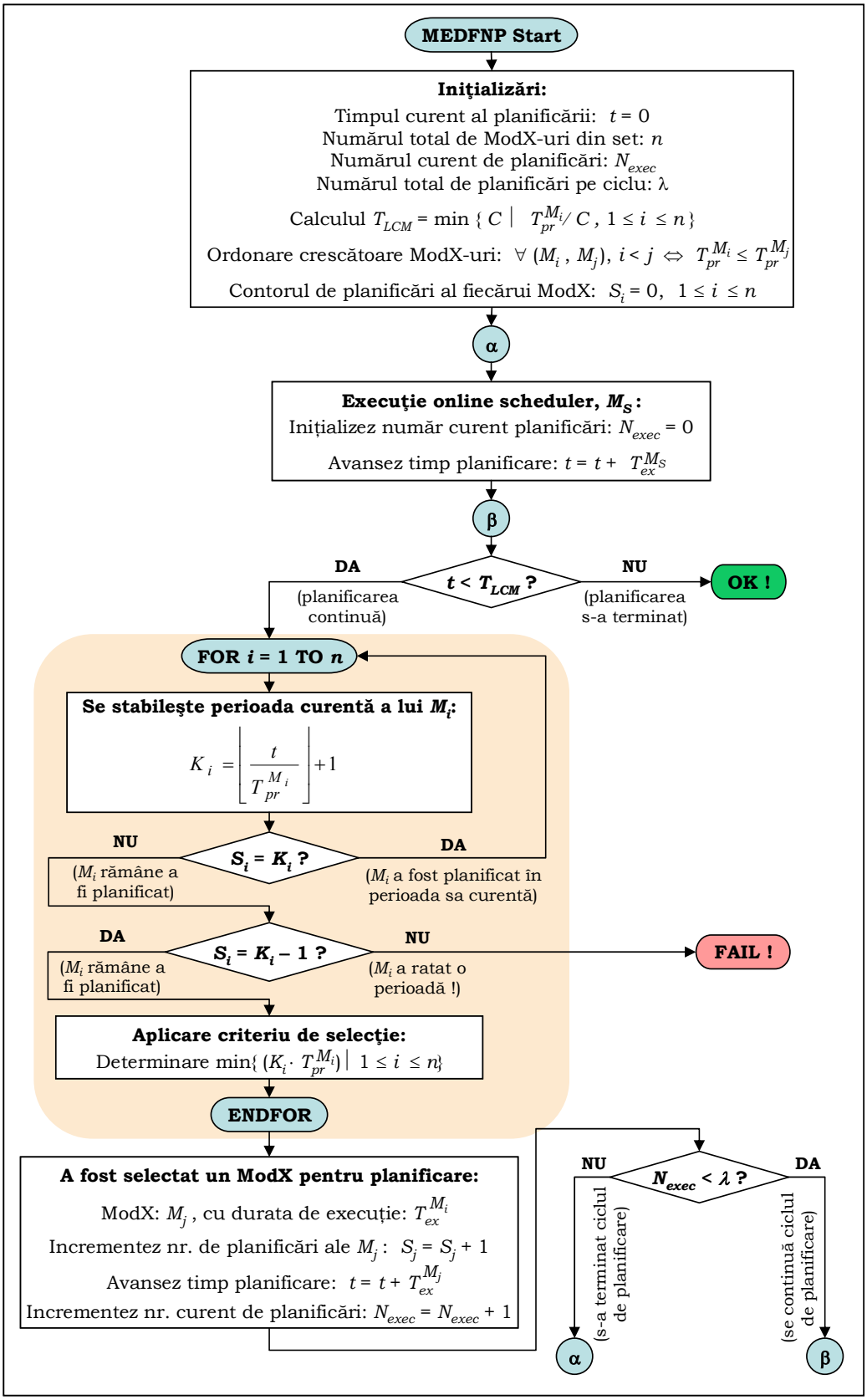

Figura 5-9. Organigrama operării de principiu a algoritmului *MEDFNP*

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

Teorema 5-1. Condiția de necesitate a planificării *MEDFNP*

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 execuție 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 țintă ce dispune de o *tabelă de dispatch* de dimensiune $\lambda + 1$ şi de un ModX special de planificare online M_S, cu durata de execuție $T_{ex}^{M_S}$.

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

$$\sum_{i=1}^{n} \frac{T_{ex}^{M_i}}{T_{pr}^{M_i}} + \left\lceil \frac{T_{LCM} \cdot \sum_{i=1}^{n} \frac{1}{T_{pr}^{M_i}}}{\lambda} \right\rceil \cdot \frac{T_{ex}^{M_S}}{T_{LCM}} \le 1 \tag{5-15}$$

Demonstrație

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

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

$$N_S = \left\lceil \frac{T_{LCM} \cdot \sum_{i=1}^{n} \frac{1}{T_{pr}^{M_i}}}{\lambda} \right\rceil \tag{5-16}$$

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

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

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:

$$U^{\boldsymbol{M}+M_S}(\infty)=U^{\boldsymbol{M}+M_S}(T_{LCM})=U^{\boldsymbol{M}}(T_{LCM})+U^{M_S}(T_{LCM})= \\ =\sum_{i=1}^{n}\frac{T_{ex}^{M_i}}{T_{pr}^{M_i}}+U^{M_S}(T_{LCM})\le 1 \tag{5-18}$$

Înlocuind pe (5-17) în (5-18), obținem condiția de necesitate a planificării *MEDFNP* specificată de (5-15).

❑

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:

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

Setul $\boldsymbol{M}$ urmează a fi planificat într-un STR cu ajutorul unui scheduler online M_S, ce dispune de o tabelă de dispatch de dimensiune 10 + 1 (λ = 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 5-1.

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 scheduler-ului online M_S pe intervalul [0, T_{LCM}) = [0, 180), se determină aplicând (5-17):

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

Utilizarea procesor a întregului sistem de task-uri, compus din setul M împreună cu ModX-ul special MS, se determină prin însumarea lui $U^{\boldsymbol{M}}(180)$ cu $U^{MS}(180)$:

$$U^{\boldsymbol{M}+MS}(180) = U^{\boldsymbol{M}}(180) + U^{MS}(180) = 0.905 \quad (5\text{-}19)$$

Din (5-19) rezultă că suma fracțiunilor din timpul procesor ocupate de execuția task-urilor planificate nu este supraunitară, deci setul $\boldsymbol{M}$ îndeplineşte condiția impusă de planificarea *MEDFNP* pe sistemul TR stabilit ca țintă.

În Figura 5-10 este reprezentată analiza offline a planificabilității setului $\boldsymbol{M}$ utilizând algoritmul *MEDFNP* pentru intervalul [0, T_{LCM}), unde T_{LCM} = 180, cu rezultate pozitive. În consecință setul $\boldsymbol{M}$ este fezabil din punct de vedere al planificării şi execuției pe sistemul țintă.

❑

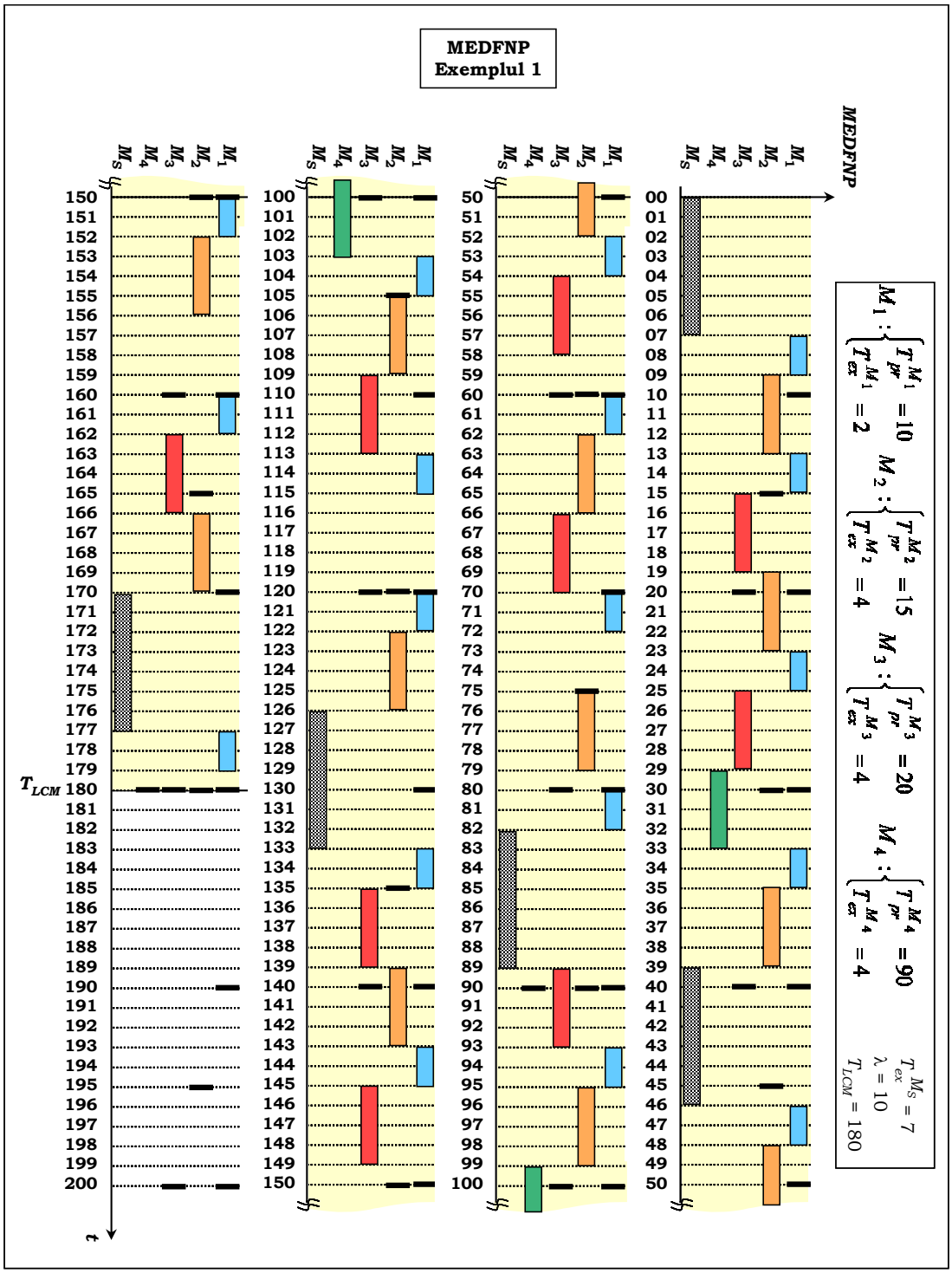

Figura 5-10. Analiza offline a planificabilității *MEDFNP* a setului ***M***

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

5.4.1 Introducere

În secțiunile anterioare ale capitolului de față a fost tratată problema planificării non-preemptive a unui set de ModX-uri simple, independente, pe sisteme TR stricte. Algoritmii de planificare discutați, dintre care *EDFNP* (respectiv *MEDFNP*) 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 specificațiile de comportare temporală.

Aşa cum rezultă din exemplele 5-1, 5-3 până la 5-6, respectiv figurile 5-3, 5-5 până la 5-7 şi 5-10, momentul exact de planificare a fiecărui ModX în cadrul perioadei sale variază de la o perioadă la alta (de la un ciclu de execuție la altul). Există însă o clasă importantă de aplicații TR stricte, şi deci un număr relativ mare de cazuri de ModX-uri, pentru care momentul execuției în cadrul fiecărei perioade trebuie să rămână constant de-a lungul operării. Exemple de aplicații cu astfel de cerințe pentru unele ModX-uri includ achizițiile periodice de date, generatoarele de semnale/de funcții programabile, sistemele de comunicație sincrone, etc.

Exemplul 5-7.

Considerăm o aplicație TR de generare a unui semnal de ieşire strict periodic, de o anumită formă. Pentru simplificare, presupunem ca necesară doar condiția ca primul eşantion al semnalului dintr-o perioadă oarecare să fie egal distanțat în timp de eşantionul corespuzător 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 execuții a task-ului trebuie să fie egale cu perioada semnalului (vezi Figura 5-11).

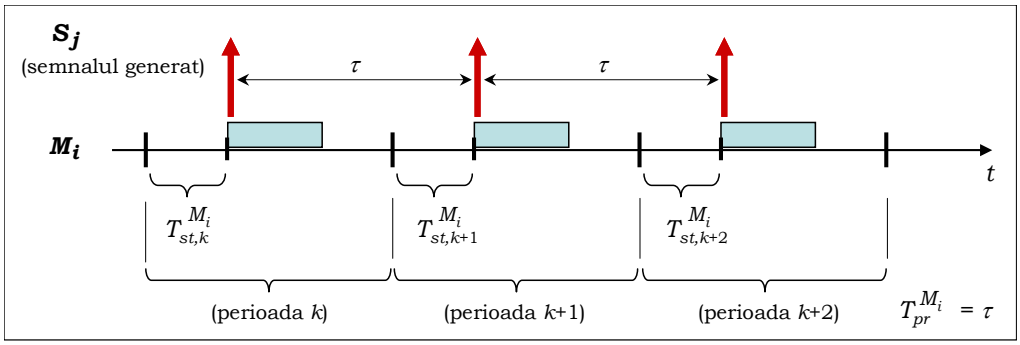

Figura 5-11. Generarea unui semnal de ieşire perfect periodic

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 execuția lui M_i, compromițând astfel generarea semnalului la perioade egale.

Utilizând modelul de ModX-uri, propus în lucrarea de față, 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 restricție în faza de specificare, anume aceea că M_i se va executa în mod obligatoriu cu aceeaşi întârziere față de momentul de început al fiecărei perioade, adică:

$$T_{st,k}^{M_i} = T_{st,k+1}^{M_i} = T_{st}^{M_i} \ , \forall k = 1,2,\ldots \tag{5-20}$$

❑

Definiția 5-3.

Numim *ModX cu execuție fixă*, sau mai simplu, *FModX*, un ModX a cărui planificare şi execuție se vor efectua în mod obligatoriu cu o aceeaşi întârziere față de momentul de început al fiecărei perioade (5-20). Cu alte cuvinte, intervalul dintre oricare două execuții 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 execuției sale este denumit *interval* (sau *durată*) *de start*, şi e notat cu $T_{st}^{M_i}$.

❑

Ca observație, menționăm că specificarea unui ModX cu execuție fixă se poate realiza şi cu ajutorul modelului de ModX descris în capitolul precedent, utilizând cei doi parametri care restricționează momentul execuției 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 execuție fixă, independente. 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 execuție fixă

Definiția 5-4.

Definim funcția $M_i(t)$ după cum urmează:

$$M_i(t): \mathsf{N} \to \{0,1\}, \text{ cu :}$$
$$M_i(t) = \sigma\left(t \bmod T_{pr}^{M_i} - T_{st}^{M_i}\right) - \sigma\left(t \bmod T_{pr}^{M_i} - T_{st}^{M_i} - T_{ex}^{M_i}\right) \quad (5\text{-}21)$$

unde: "mod" reprezintă *operatorul modulo*, iar $\sigma : \mathsf{Z} \to \{0,1\}$, $\sigma(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$ este *funcția treaptă unitate*.

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

❑

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

Exemplul 5-8.

Considerăm un ModX cu execuție fixă M_1, caracterizat prin următorii parametri temporali (Figura 5-12):

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

Operarea FModX-ului M_1 este modelată de funcția:

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

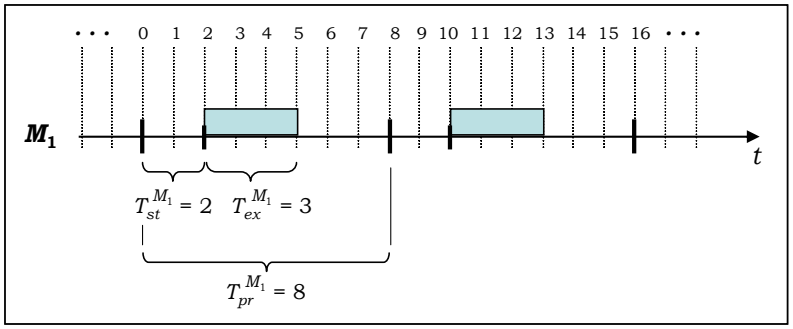

Figura 5-12. Operarea FModX-ului M1 pentru (primele) două perioade ale sale

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

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

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:
$$\begin{cases} (0+a) \bmod n = a \bmod n \\ (1 \cdot a) \bmod n = a \bmod n \end{cases},\ \forall a \in Z_n .$$

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 față de adunare şi înmulțire:

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

Proprietatea 5-9. Congruența 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 execuție fixă, ordonate crescător după perioadă:

$$\begin{cases}M_1\equiv\left(T_{pr}^{M1},T_{ex}^{M1},T_{st}^{M1}\right)\\M_2\equiv\left(T_{pr}^{M2},T_{ex}^{M2},T_{st}^{M2}\right)\end{cases},\ \text{cu } T_{pr}^{M1}\le T_{pr}^{M2}.$$

Conform cu Definiția 5-4, cele două FModX-uri sunt modelate cu ajutorul următoarelor funcții:

$$\begin{cases}M_1(t)=\sigma\left(t\bmod T_{pr}^{M1}-T_{st}^{M1}\right)-\sigma\left(t\bmod T_{pr}^{M1}-T_{st}^{M1}-T_{ex}^{M1}\right)\\M_2(t)=\sigma\left(t\bmod T_{pr}^{M2}-T_{st}^{M2}\right)-\sigma\left(t\bmod T_{pr}^{M2}-T_{st}^{M2}-T_{ex}^{M2}\right)\end{cases},\ t\in\mathsf{N}$$

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

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

$$\mathcal{D}\overset{\text{not}}{=}\text{GCD}\left\{T_{pr}^{M1},T_{pr}^{M2}\right\}=\max\left\{\delta\in\mathsf{N}^*\ \middle|\ T_{pr}^{M1}\bmod\delta=T_{pr}^{M2}\bmod\delta=0\right\}\quad(5\text{-}22)$$

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

$$\mathcal{M}\overset{\text{not}}{=}\text{LCM}\left\{T_{pr}^{M1},T_{pr}^{M2}\right\}=\min\left\{\mu\in\mathsf{N}^*\ \middle|\ \mu\bmod T_{pr}^{M1}=\mu\bmod T_{pr}^{M2}=0\right\}\quad(5\text{-}23)$$

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:

$$\Delta_{M_2/M_1} : \left\{0,1,\ldots,T_{pr}^{M_2}-1\right\} \to \{0,1\}$$
$$\Delta_{M_2/M_1}(\tau) = \bigcup_{k=0}^{\frac{T_{pr}^{M_1}}{\mathcal{D}}-1} M_1\left(\tau + k \cdot T_{pr}^{M_2}\right) \tag{5-24}$$

❑

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 "$\bigcup$" – "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 5-9.

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

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

Cei doi parametri suplimentari ce derivă din caracteristicile temporale ale setului $\boldsymbol{M}$ sunt, conform (5-22) şi (5-23):

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

adică, cel mai mare divizor comun, respectiv cel mai mic multiplu comun al perioadelor lui M_1 şi M_2. Conform relației (5-24), funcția de mapare a planificării fixe a lui M_1 în perioada lui M_2 este de forma:

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

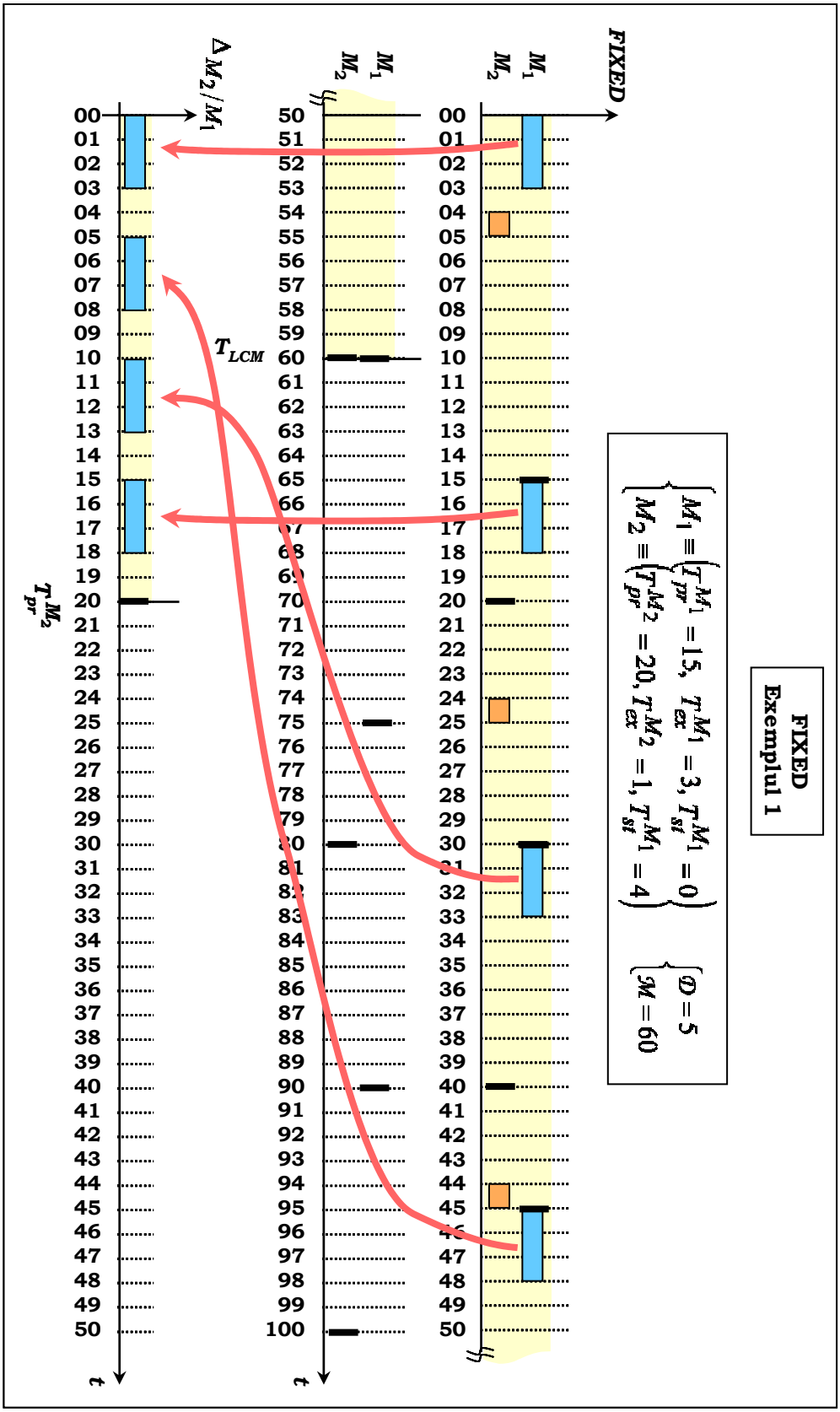

Figura 5-13. Planificarea fixă a setului M de FModX-uri şi funcția de mapare

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 5-13 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 (5-24) şi (5-21):

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

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

❑

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 5-2.

Fie $\boldsymbol{M} = \{M_1, M_2\}$, un set de două ModX-uri cu execuție 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 $\mathcal{D} \in \mathsf{N}^*$, cel mai mare divizor comun al perioadelor lui M_1 şi M_2:

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

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

$$\Delta_{M_2/M_1}(t_0) = \Delta_{M_2/M_1}(t_0 + \mathcal{D}), \;\; \forall t_0 \in \left\{0,1,\ldots,T_{pr}^{M_2} - 1\right\} \tag{5-25}$$

Demonstrație

Din definiția funcției de mapare (5-24) şi din (5-25), va trebui demonstat că:

$$\begin{gathered} \forall k \in \left\{0,1,\ldots,\frac{T_{pr}^{M_1}}{\mathcal{D}} - 1\right\}, \; \exists m \in \left\{0,1,\ldots,\frac{T_{pr}^{M_1}}{\mathcal{D}} - 1\right\}, \text{ astfel incat :} \\ M_1\left(t_0 + k \cdot T_{pr}^{M_2}\right) = M_1\left(t_0 + \mathcal{D} + m \cdot T_{pr}^{M_2}\right) \end{gathered} \tag{5-26}$$

Introducem notațiile:

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

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

$$\beta \overset{\text{not}}{=} \gamma \cdot \alpha + \delta \tag{5-27}$$

unde: $\delta \in Z_\alpha = \{0, 1, \ldots, \alpha - 1\}$, iar α şi δ sunt relativ prime ($\text{GCD}\{\alpha, \delta\} = 1$). Relația (5-27) se obține din faptul că $\text{GCD}\left\{T_{pr}^{M_1}, T_{pr}^{M_2}\right\} = \mathcal{D}$. Presupunând că $\text{GCD}\{\alpha, \delta\} = \xi > 1$, ar rezulta că $\text{GCD}\left\{T_{pr}^{M_1}, T_{pr}^{M_2}\right\} = \mathcal{D} \cdot \xi$, fals.

Cu notațiile de mai sus şi aplicând definiția funcției $M_1(t)$ conform (5-21), formula (5-26) ce trebuie demonstrată devine:

$$\begin{gathered} \forall k \in \mathsf{Z}_\alpha = \{0,1,\ldots,\alpha - 1\}, \; \exists m \in \mathsf{Z}_\alpha, \text{ astfel incat :} \\ (t_0 + k \cdot \beta \cdot \mathcal{D}) \bmod (\alpha \cdot \mathcal{D}) = (t_0 + (m \cdot \beta + 1) \cdot \mathcal{D}) \bmod (\alpha \cdot \mathcal{D}) \end{gathered} \tag{5-28}$$

Aplicând Proprietatea 5-9 a operatorului "mod", (5-28) 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, (5-27) permite existența elementului invers față de înmulțirea modulară (Proprietatea 5-7):

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

Rezultă:

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

$$m = (k - \mu) \bmod \alpha \qquad (5\text{-}29)$$

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 (5-26) este adevărată. Mai mult, m ia toate valorile din mulțimea $Z_\alpha = \{0, 1, \ldots, \alpha - 1\}$, pe măsură ce şi k baleiază întreagul Z_α.

❑

Corolarul 5-1.

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

$$\nu_{M_2/M_1} = \frac{T_{pr}^{M_2}}{\mathcal{D}} \qquad (5\text{-}29)$$

Demonstrație

Demonstrația este o consecință imediată a definiției funcției $\Delta_{M_2/M_1}(\tau)$ (Definiția 5-5) şi a Teoremei 5-2. Astfel, ştim că $\Delta_{M_2/M_1}(\tau)$ este definită pentru $\tau \in \{0,1,\ldots,T_{pr}^{M_2} - 1\}$, adică pentru $T_{pr}^{M_2}$ unități de timp, şi este o funcție 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 observație, menționăm că pe durata unui interval de planificare ($T_{LCM} = \mathrm{LCM}\left\{T_{pr}^{M_1}, T_{pr}^{M_2}\right\} = \mathcal{M}$, conform relației (5-23)), se vor mapa în perioada lui M_2 exact

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

execuții ale lui M_1. De aici şi din Corolarul 5-1 rezultă că funcția de mapare a planificării fixe a lui M_1 în perioada lui M_2 este o funcție definită pe $T_{pr}^{M_2}$ instanțe 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 execuție a lui M_1, de durată $T_{ex}^{M_1}$.

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

Teorema 5-3.

Fie $\boldsymbol{M} = \{M_1, M_2\}$, un set de două ModX-uri cu execuție 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:

$$\exists t_0 \in \left\{0,1,\ldots,T_{pr}^{M_2} - T_{ex}^{M_2}\right\}, \text{ astfel incat } \bigcup_{\tau=t_0}^{t_0+T_{ex}^{M_2}-1} \Delta_{M_2/M_1}(\tau) = 0 \qquad (5\text{-}30)$$

Demonstrație

a) Implicația directă: dacă (5-30) e adevărată, rezultă că setul $\boldsymbol{M}$ este fezabil.

Dacă (5-30) 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 execuția 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 execuției lui M_2 în cadrul planificării setului $\boldsymbol{M}$.

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

b) Implicația inversă: dacă setul $\boldsymbol{M}$ este fezabil, rezultă că (5-30) se verifică.

Dacă setul $\boldsymbol{M}$ este fezabil, rezultă că există o planificare a lui M_1 şi M_2 astfel încât execuțiile 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 (5-30) 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 5-3.

❑

Corolarul 5-3.

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

$$T_{ex}^{M_1} + T_{ex}^{M_2} \le \mathcal{D} = \text{GCD}\left\{T_{pr}^{M_1}, T_{pr}^{M_2}\right\} \tag{5-31}$$

Demonstrație

Teorema 5-3 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:

$$T_x \ge T_{ex}^{M_2} \tag{5-32}$$

Pe de altă parte, conform Teoremei 5-2, 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 aceleiași 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:

$$T_x \le \mathcal{D} - T_{ex}^{M_1} \tag{5-33}$$

Din (5-32) şi (5-33) rezultă că relația (5-31) din enunț se verifică.

❑

Corolarul 5-3 exprimă practic, *condițiile de necesitate şi suficiență* ale planificării non-preemptive pentru un set de două ModX-uri independente, cu execuție fixă:

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

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

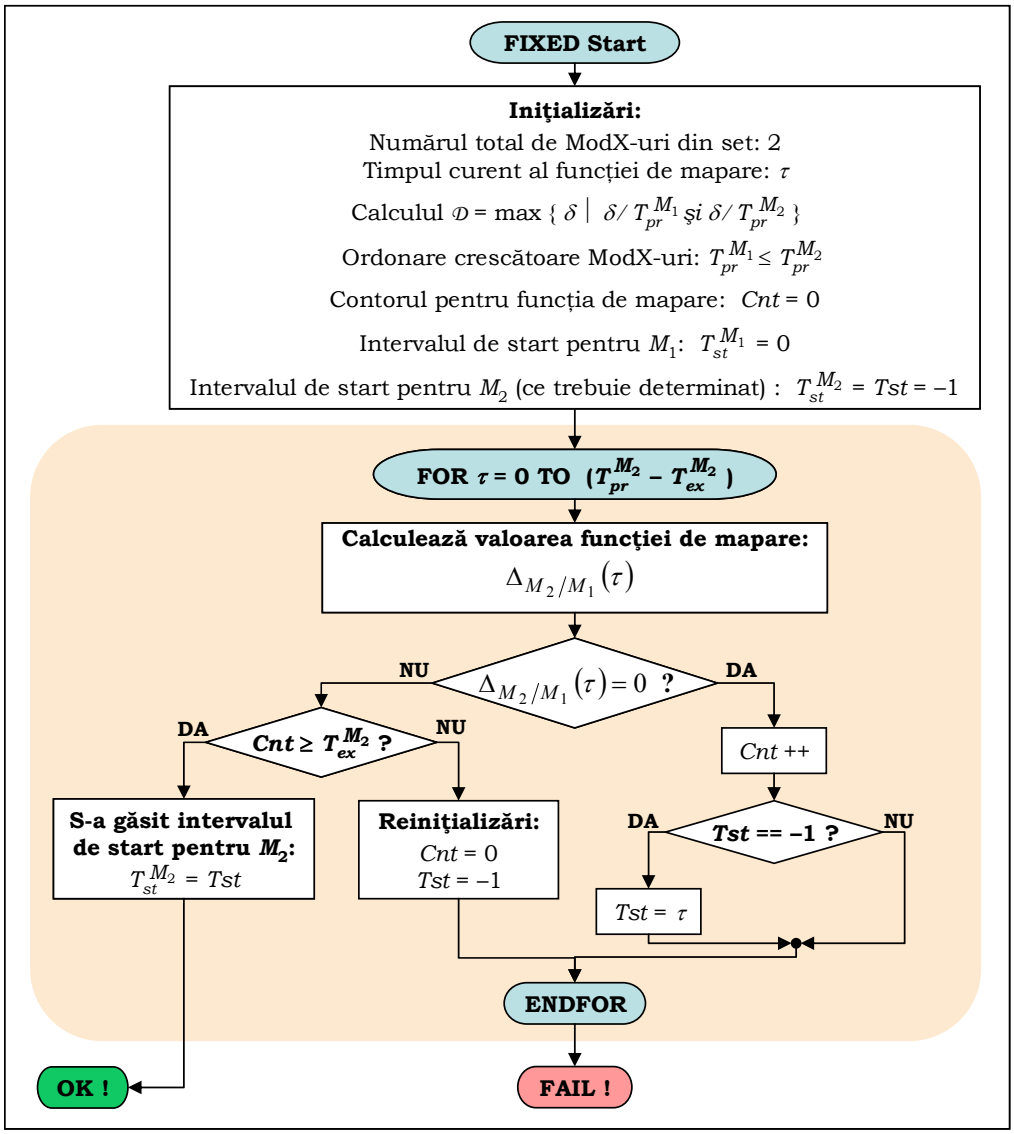

Figura 5-14. Algoritmul de planificare non-preemptivă a unui set de două FModX-uri

Algoritmul de planificare non-preemptivă a două FModX-uri independente (Figura 5-14), 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 5-9, algoritmul va obține $T_{st}^{M_2} = 3$, rezultând într-o planificare fezabilă a setului (alta decât cea ilustrată în Figura 5-13). Exemplul următor prezintă toate procedurile implicate în analiza offline a planificabilității setului de FModX-uri considerat.

Exemplul 5-10.

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

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

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 (5-22) şi (5-23), cel mai mare divizor comun, respectiv, cel mai mic multiplu comun al perioadelor FModX-urilor, sunt:

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

Primul pas constă din aplicarea condiției de necesitate şi suficiență a planificabilității non-preemptive a setului $\boldsymbol{M}$, conform (5-31):

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

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 5-15 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.

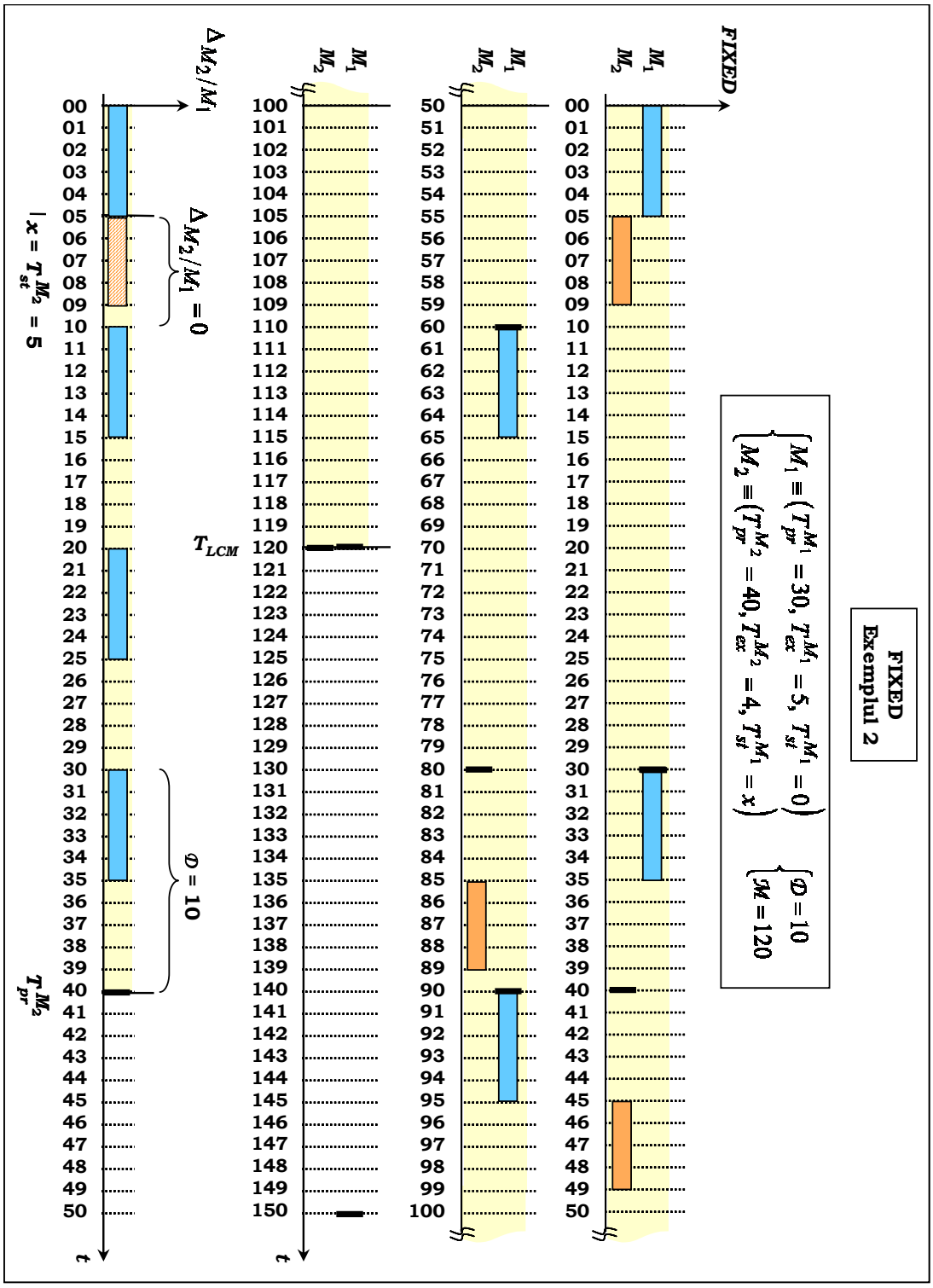

Figura 5-15. Analiza offline a planificării setului ***M***

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 definește 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 5-15).

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

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

❑

Generalizând principiile de bază prezentate şi demonstrate în această secțiune, 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. Acesta este unul din subiectele prevăzute a fi abordate în continuare în cadrul activității noastre de doctorat şi va fi prezentat în teză.

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 restricții, plecând de la situațiile mai simple (şi mai aproape de ideal) şi avansându-se către situații mai complexe şi mai realiste.

Pentru seturile de ModX-uri independente au fost introduşi şi studiați doi dintre cei mai eficienți 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. În acelaşi context, a fost abordată problema evaluării fezabilității setului de task-uri utilizând condiții de necesitate şi/sau suficiență. Am demonstrat că pentru modelul de task-uri considerat (ModX-uri independente, cu perioadele aliniate la momentul inițial t_0 = 0), condiția a doua de necesitate enunțată 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 aplicației, şi un task special – planificatorul online, ce utilizează o tabelă de dispatch de dimensiuni fixe, bine determinate, pentru planificarea ModX-urilor în cicluri cu număr constant de

execuții. Ținând cont şi de caracteristicile de comportare temporală a planificatorului online, au fost concepute şi demonstrate condițiile de necesitate ale planificabilității sistemului. Algoritmul de planificare non-preemptivă pentru astfel de sisteme de task-uri a fost prezentat şi verificat prin simulări şi exemple.

În fine, am introdus şi abordat problema modelării şi planificării seturilor de task-uri ce necesită o execuție fixă în cadrul fiecărei perioade a acestora. A fost conceput şi studiat modelul matematic al ModX-urilor cu execuție fixă (FModX-uri). A fost analizată planificarea non-preemptivă a seturilor compuse din două FModX-uri independente, prin introducerea şi demonstrarea condițiilor de necesitate şi suficiență ale planificabilității, şi prin conceperea algoritmului de planificare ce utilizează funcția de mapare. Modul de operare al algoritmului a fost exemplificat pe diverse cazuri.

Preocupările curente şi de viitor imediat din cadrul activității noastre de doctorat cuprind următoarele subiecte legate de planificarea non-preemptivă a seturilor de ModX-uri, ce necesită încă a fi rezolvate:

- Planificarea seturilor compuse dintr-un număr oarecare de ModX-uri cu execuție fixă;
- Planificarea seturilor de ModX-uri cu dependențe de program (cu alte cuvinte, planificarea unei aplicații TR complete);
- Integrarea tuturor cazurilor de planificare non-preemptivă şi conceperea unui model unificat, capabil a fi utilizat în cât mai multe aplicații reale.

6 Modelul STR strict pentru aplicații critice APNS şi de control încorporat

În Capitolul 3 au fost discutate problemele actuale şi cerinţele hardware şi software impuse de sistemele timp-real destinate aplicațiilor critice, vizând, pentru o mare parte a task-urilor acestora, respectarea cu strictețe a termenelor specificate.

Un model temporal pentru dezvoltarea STR stricte a fost propus în capitolul următor, şi, pe baza acestuia a fost definit şi analizat un set minimal dar suficient de proprietăți ce definesc comportarea semnalelor în timp. Utilizând acelaşi model temporal şi specificațiile semnalelor cu care interacționează sistemele TR, au fost analizate proprietățile task-urilor, în particular, al task-urilor TR stricte, pentru care respectarea termenelor impuse în faza de specificare reprezintă o cerință esențială. Ca rezultat, a fost introdus un model de task TR strict – ModX-ul, care, prin concepția modulară şi prin setul de parametri temporali, oferă o soluție viabilă pentru dezvoltarea de sisteme şi aplicații TR stricte.

Capitolul 5 tratează problema esențială a planificării task-urilor unui STR strict. În particular, utilizând modelele pentru timp, pentru semnale şi task-uri propuse în capitolul anterior, se abordează problema planificării non-preemptive a seturilor de ModX-uri.

Unificarea acestor modele şi caracteristici într-un sistem omogen este prezentată în capitolul de față. Aici, vom introduce un model de STR conceput pentru a oferi un mediu integrat, unitar, de specificare, programare, analiză şi execuție a aplicațiilor critice din domenii ca achiziția şi prelucrarea numerică a semnalelor, orientat pe arhitecturi compacte, de control digital încorporat, cum ar fi sistemele bazate pe procesoare numerice de semnal, de exemplu.

6.1 Descrierea generală a sistemului OPEN-HARTS

Una din observațiile esențiale desprinse din studiul sistemelor TR stricte şi a algoritmilor de planificare a ModX-urilor este că faza de analiză offline, apriori execuției în sistem a setului de task-uri TR stricte, reprezintă o cerință obligatorie. În consecință, modelul STR strict conceput cuprinde două elemente principale: un mediu integrat, destinat dezvoltării şi analizei offline a aplicațiilor şi un nucleu de operare TR strict pe care se încarcă şi se execută aplicațiile validate.

OPEN-HARTS (Operating Environment for Hard Real-Time Systems) este conceput ca un mediu de operare pentru sisteme şi aplicații TR stricte, având două componente majore (vezi Figura 6-1):

1) Mediul integrat de dezvoltare şi analiză a aplicațiilor 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 aplicații TR un mediu vizual integrat, destinat proiectării, specificării şi verificării formale a aplicațiilor (cu accent pe comportamentul temporal al task-urilor – obligatoriu la cele TR stricte). INVERTA permite de asemenea, programarea funcțională a fiecărui task al aplicației, analiza temporală a task-urilor (extragerea WCET) şi analiza planificabilității. Toate operațiile 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 țintă, şi permite încărcarea şi execuția task-urilor TR stricte (Hard Real-Time tasks) în regim non-preemptiv şi a celor lejere (Soft Real-Time tasks) în regim preemptiv.

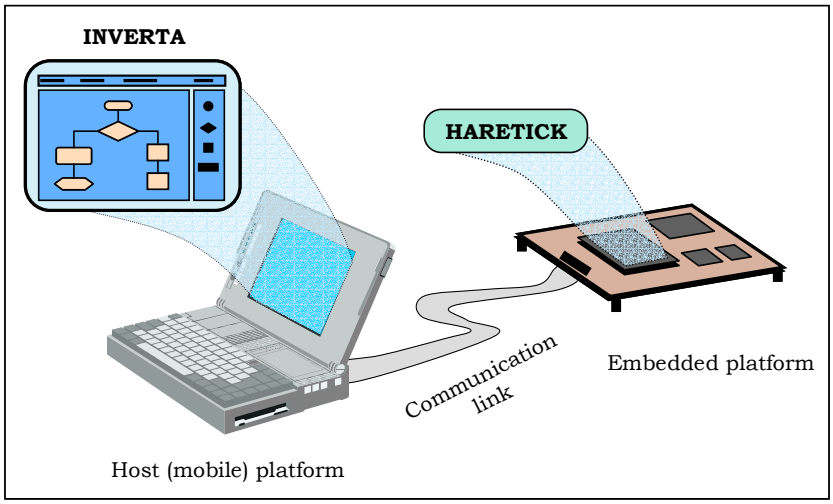

Figura 6-1. Arhitectura generală a sistemului OPEN-HARTS

Cele două componente ale sistemului OPEN-HARTS sunt interconectate printr-o legătură şi un protocol de comunicații, permițând schimbul de date, comenzi şi informații de stare între INVERTA (programatorul de sistem/aplicații) şi HARETICK (sistemul TR care operează pe o platformă țintă).

6.2 Caracteristici generale

Întregul sistem se bazează pe o concepție puternic modulară, atât în ceea ce priveşte arhitectura generală cât şi modul de abordare a proiectării, specificării, analizei şi execuției aplicațiilor TR.

Mediul integrat INVERTA permite programatorului de sistem/aplicații dezvoltarea unei aplicații TR în regim offline, fără a interfera deci, în această fază, cu sistemul țintă, pe care poate opera la un moment dat o versiune anterioară a aplicației. Dezvoltarea aplicației 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 rețea vizuală, echivalentă cu graful dependențelor 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 (dependența de date față de celelalte module).
- Setul variabilelor globale ale aplicației, utilizate de către task.
- Setul semnalelor de intrare şi de ieşire cu care interacționează 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 funcțională a fiecărui task se va specifica prin intermediul unui limbaj de programare specific aplicațiilor TR (limbaj de asamblare, C modificat, etc.).

După proiectarea şi specificarea aplicației, INVERTA va trece în mod automat la faza de verificare/validare a specificațiilor. Sistemul va utiliza seturi de formule bine stabilite, între parametrii temporali ai semnalelor şi ai task-urilor aplicației (prezentate în Capitolul 4), atât pentru a verifica validitatea specificațiilor, cât şi pentru a stabili în mod automat valorile parametrilor ce nu au fost specificați explicit de programator. INVERTA verifică, de asemenea, dependențele de control şi de date ale aplicației, ținând cont şi de specificarea funcțională (codul program) al fiecărui task, şi modifică acolo unde e cazul, afişarea acestor dependențe în mod corespunzător.

După validarea întregii aplicații, atât din punct de vedere funcțional, cât şi al comportamentului în timp, INVERTA va compila codurile sursă ale fiecărui task şi va efectua analiza temporală a programelor, pentru obținerea timpilor maximi de execuție pentru fiecare task (*WCET*). Tot în cadrul fazei de analiză a aplicației, INVERTA aplică algoritmii planificare special concepuți pentru modelul de sisteme TR stricte considerate în lucrare (Capitolul 5), pentru a testa planificabilitatea setului de ModX-uri ale aplicației.

În cazul în care INVERTA a găsit o planificare fezabilă a aplicației, se efectuează pregătirile finale pentru încărcarea ei pe platforma țintă, unde va fi executată sub nucleul de operare HARETICK: link-editare, generarea tabelei de procese (ce conține şi parametrii temporali ai task-urilor) şi a grafului aplicației, generarea tabelelor de simboluri şi a hărții de memorie, etc.

Legătura de date şi comenzi (DATALINK) dintre INVERTA şi HARETICK are următoarele caracteristici principale:

- Implementează un protocol de comunicații simplu, dar flexibil şi scalabil.
- Permite transmiterea de comenzi utilizator prin intermediul INVERTA, către nucleul de operare HARETICK, care execută o aplicație TR oarecare.

- Permite citirea stării de operare a sistemului țintă, prin comandarea generării de diverse rapoarte de execuție.
- Permite încărcarea unei aplicații noi (sau modificate) în sistemul țintă, sau descărcarea informațiilor complete despre o aplicație ce se execută pe sistemul țintă. Datele ce caracterizează execuția unei aplicații TR oarecare pe platforme HARETICK, cuprind printre altele, următoarele elemente:
  - sursele de cod compilat ale task-urilor aplicației;
  - tabela cu descriptorii de procese (incluzând parametrii temporali ai task-urilor);
  - tabela ce descrie graful aplicației;
  - harta de memorie;
  - tabela de simboluri.

Aşadar, prin intermediul legăturii de date, a protocolului de comuncație şi a interfețelor corespunzătoare din INVERTA şi HARETICK, programatorul de aplicații poate să citească la orice moment starea sistemului țintă (în timpul operării acestuia, şi fără a afecta operarea sistemului) – incluzând informațiile complete despre aplicația executată curent, poate să trimită diverse comenzi utilizator către sistem şi să încarce o nouă aplicație pentru a o înlocui pe cea curentă.

La terminarea cu succes a încărcării unei aplicații în cadrul sistemului țintă, nucleul de operare HARETICK va începe execuția acesteia în timp real. Acest moment defineşte practic începerea operării efective a aplicației în cadrul sistemului, şi predictibilitatea execuției, asupra căreia se concentrează materialul din lucrarea de față, 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 execuție aplicațiilor TR critice. Caracteristica de nucleu TR hibrid se referă la faptul că HARETICK oferă mecanisme pentru existența a două regimuri "simultane" de execuție: contextul non-preemptiv, destinat execuției task-urilor TR stricte (ModX-urile), cu garantarea respectării termenelor specificate (oferind deci, o predictibilitate maximă), şi contextul preemptiv destinat execuției task-urilor lejere din punct de vedere al specificațiilor de comportament temporal. Mecanismele implementate se bazează pe limitarea (sau chiar eliminarea) întreruperilor din sistem, cu excepția întreruperii de ceas (*RTC – Real-Time Clock*), care are prioritatea maximă.

Observație

Modelul kernelului hibrid de operare TR descris mai sus a fost propus şi introdus în contextul implementării HARETICK, ca soluție (cel puțin parțială) la problema eficienței 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 execuție" (WCET) a task-urilor sistemului.

Conceptele de arhitectură şi implementare introduse în dezvoltarea nucleului TR hibrid HARETICK constituie subiectul unei lucrări viitoare.

6.3 Principiile de operare ale sistemului propus

Principiile de operare ale sistemului OPEN-HARTS au fost prezentate în esență în paragraful anterior. Pentru o exemplificare şi mai concretă a capabilităților sistemului şi a facilităților pe care acesta le pune la dispoziția programatorului de sistem/aplicații TR, 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 cerințe critice de operare. Pe platforma sistemului rulează nucleul HARETICK, ce execută o anume aplicație TR. Programatorul sistem doreşte modificarea aplicației care rulează în mod curent pe platforma țintă, mai precis, doreşte adăugarea a încă unui ModX în aplicație, de exemplu. Secvența 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 țintă, 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. Comada 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 a sistemului țintă 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 înărcării cu succes pe platforma țintă a tuturor datelor necesare planificării şi executării online a noii aplicații. După acest moment, aplicația va opera în timp-real, cu garantarea respectării tuturor specificațiilor temporale pentru task-urile TR stricte (ModX-urile aplicației).

De remarcat faptul că, până la terminarea pasului 10 din secvența de mai sus, cu alte cuvinte, până când noua aplicație este complet încărcată în sistemul țintă, acesta va continua să execute aplicația veche. Comutarea execuției dintre cele două aplicații 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 interacțiunea sistemului cu mediul înconjurător.

7 Concluzii

7.1 Concluzii

Referatul de față face parte din programul de doctorat cu tema "Contribuții la achiziția şi prelucrarea numerică în timp-real a semnalelor utilizând sisteme de calcul distribuite", sub coordonarea ştiințifică a prof. dr. ing. Vladimir CREȚU.

În cadrul lucrării a fost abordată problematica sistemelor TR stricte destinate aplicațiilor critice de achiziție şi prelucrare numerică a semnalelor (DSP-based systems) şi de control digital încorporat (embedded systems). În Capitolul 3 au fost prezentate principalele cerințe impuse de către astfel de sisteme TR stricte:

- introducerea timpului ca şi coordonată esențială a operării sistemului;
- modelarea corespunzătoare a comportamentului temporal al semnalelor şi task-urilor sistemului;
- definirea clară a mai multor faze necesare în dezvoltarea sistemelor/aplicațiilor TR: specificarea, proiectarea, verificarea şi validarea, şi analiza fezabilității. Metodele cu ajutorul cărora se abordează aceste faze trebuie să fie unitare;
- asigurarea predictibilității execuției task-urilor TR stricte ale sistemului, reprezintă o cerință esențială, ce afectează toate etapele dezvoltării şi implementării sistemului;
- algoritmii de planificare pentru STR sunt special concepuți, trebuind să asigure execuția predictibilă, cu respectarea termenelor impuse task-urilor TR stricte, conferind sistemului, pe de altă parte, eficiență şi flexibilitate cât mai mari;
- programarea task-urilor aplicațiilor TR trebuie realizată cu ajutorul unor limbaje care să permită exprimarea de constrângeri temporale şi totodată să permită analiza temporală a programelor (calculul timpilor WCE, BCE, etc.).

Cu toate că domeniul sistemelor TR a beneficiat de o atenție deosebită din partea comunității academice şi industriale în ultimele decenii, există încă o serie de probleme nerezolvate (Capitolul 3). Majoritatea sistemelor TR (în special cele stricte) dezvoltate şi utilizate în prezent reprezintă practic implementări ad-hoc, puternic orientate pe aplicațiile vizate şi particularizate pentru arhitecturile specifice implementării. Nu există încă un model de sistem TR cu aplicabilitate mai largă (pentru un număr şi o varietate mai mare de aplicații TR) şi care să fie general acceptat în domeniu.

De asemenea, încă se practică pe scară largă, proiectarea şi dezvotarea STR prin adaptarea/modificarea unor sisteme de operare de uz general (caracterizate prin comportare optimă, eficientă, pentru situațiile comune de operare – încărcare medie a

sistemului, etc. – dar care nu se comportă predictibil şi corect din punct de vedere temporal în toate cazurile, de exemplu, în condițiile cele mai defavorabile de operare). Această abordare este aplicată atât sub aspect hardware (utilizarea memoriei chache, pipelining, speculative computing, memoria virtuală, utilizarea extensivă a întreruperilor, etc.) cât şi software (bucle infinite, referințe şi structuri de date dinamice, recursivitatea, întârzieri implementate ca bucle goale, etc.).

Numeroase studii şi modele au fost realizate şi propuse, în special în mediul academic, pentru utilizarea în cadrul STR. Acestea însă abordează în general un set de probleme punctuale, nereuşind să rezolve tot spectrul de probleme ce apar în cadrul operațiilor de dezvoltare a STR, de la stadiul de specificare şi până la testarea şi evaluarea implementării. Mai mult, majoritatea acestor modele sunt concepute cât mai aproape de modele ideale, impunând în consecință restricții majore în cazul implementărilor practice, concrete. Astfel, există un număr relativ mare de studii asupra modelelor temporale şi utilizarea acestora în specificarea şi verificarea formală a STR. Modelele temporale nu sunt însă integrate şi în limbaje de programare corespunzătoare, sau în tehnicile de planificare a task-urilor.

Algoritmii de planificare a seturilor de task-uri ale STR constituie un subiect de interes major în domeniul STR. În literatura de specialitate au fost propuşi şi studiați un număr mare de algoritmi şi tehnici de planificare, dar, şi de această dată, majoritatea studiilor au fost efectuate asupra unor modele de seturi de task-uri apropiate de ideal, ce impun de regulă constrângeri dificil de rezolvat din punct de vedere practic. Pe de altă parte, din cauză că majoritatea algoritmilor sunt concepuți pentru planificarea task-urilor în regim preemptiv, aceştia nu pot garanta în toate cazurile respectarea constrângerilor temporale, mai ales pentru seturi de task-uri TR stricte destinate aplicațiilor critice.

În lucrarea de față am abordat problema sistemelor TR stricte în ansamblul ei, considerând toate etapele dezvoltării acestora: specificare şi verificare formală, programare, analiză a fezabilității, implementare şi testare.

Ideea principală a lucrării se bazează pe observația că utilizarea întreruperilor în sistemele TR stricte reprezintă o problemă din punctul de vedere al predictibilității, afectând capacitatea acestora de a garanta în orice condiții de operare respectarea termenelor impuse task-urilor TR stricte (Capitolul 5).

A fost analizat şi propus un model temporal (Capitolul 4), urmând a se constitui într-un element de bază pentru toate fazele dezvoltării STR stricte. Următoarele elemente principale caracterizează modelul temporal:

- domeniul temporal are o structură liniară, discretă şi limitată la stânga (existența momentului inițial, care corespunde startării STR);
- domeniul temporal poate fi echivalat cu mulțimea numerelor naturale;
- unitatea de timp are o corespondență 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 relații cantitative între instanțe individuale de timp (puncte) sau între intervale temporale;
- metrica pentru timp permite determinarea lungimii (duratei) intervalelor temporale;
- există o relație bine stabilită între domeniul temporal sistem şi cel absolut.

Task-urile sistemului sunt privite ca seturi de acțiuni în strânsă relație de cauzalitate cu evenimentele (semnalele): apariția unui eveniment (sau a unei secvențe bine definite de evenimente) provoacă o acțiune şi reciproc. În prima situație, e vorba de task-uri ce *tratează* semnale de intrare, iar în a doua situație, de task-uri ce *generează* semnale de ieşire. Comportarea în timp a unui task TR generic a fost studiată şi modelată cu ajutorul unui set de parametri şi a relațiilor stabilite între aceştia (Tabelul 4-3 şi relațiile (4-9) până la (4-15) din Capitolul 4). În continuare, au fost evidențiate avantajele şi problemele ridicate de tratarea semnalelor de intrare cu ajutorul task-urilor periodice (utilizând tehnica de *polling*). O soluție la această problemă, formulată la nivelul specificării şi verificării formale a semnalelor şi a task-urilor, a fost prezentată şi demonstrată cu ajutorul Teoremei 4-1. Utilizarea task-urilor periodice pentru tratarea semnalelor de intrare garantează respectarea termenelor stricte, însă în detrimentul eficienței de operare (Exemplul 4-6). Soluții ce vizează creşterea eficienței sunt propuse şi exemplificate în Capitolul 5 şi 6.

Sintetizând studiul semnalelor şi al task-urilor pe baza modelului temporal considerat, a fost introdus şi discutat modelul task-ului TR strict, ModX-ul, care înglobează aspectele considerate de importanță majoră în dezvoltarea, analiza şi implementarea STR stricte:

- Task-urile periodice, executate în context non-preemptiv oferă predictibilitate maximă sistemelor TR.
- Abordarea modulară uşurează concepția, specificarea şi proiectarea aplicațiilor.
- Sub aspect funcțional, este necesară impunerea de restricții la programarea ModX-urilor: evitarea recursivității, limitarea în timp şi spațiu a buclelor de program, etc. Obiectivul major îl reprezintă facilizarea analizei temporale de program şi determinarea timpului maxim de execuție a modulului (WCET).
- Comunicația între modulele aplicației se realizează prin intermediul parametrilor de intrare/ieşire şi, într-o manieră limitată, prin variabilele globale. Datorită execuției ModX-urilor în context non-preemptiv, acestea implementează practic operații 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 relații directe de legătură cu parametri similari ai semnalelor. Capitolul 4 tratează în detaliu şi acest aspect.

Odată cu modelul task-ului TR strict, au fost introduse şi două noțiuni relativ noi. "Contorul de execuție" al ModX-urilor este un parametru ce specifică şi controlează numărul ciclurilor (perioadelor) lor de execuție. Valoarea contorului de execuție poate fi setată pe de o parte de către programatorul de sistem/aplicație (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.

Problema planificării seturilor de task-uri TR a fost abordată 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). Avantajele şi dezavantajele utilizării algoritmilor de planificare non-preemptivă au fost discutate la începutul Capitolului 5. 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).

Au fost considerate mai multe cazuri, pentru care s-au aplicat diverse restricții, plecând de la situațiile mai simple (şi mai aproape de ideal) şi avansându-se către situații mai complexe şi mai realiste. Cu ajutorul Lemei 5-1 am 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 şi studiați doi dintre cei mai eficienți 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. În acelaşi context, a fost abordată problema evaluării fezabilității setului de task-uri utilizând condiții de necesitate şi/sau suficiență. Am demonstrat că pentru modelul de task-uri considerat (ModX-uri independente, cu perioadele aliniate la momentul inițial $t_0 = 0$), condiția a doua de necesitate enunțată 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 aplicației, şi un task special – planificatorul online, care utilizează o tabelă de dispatch de dimensiuni fixe, bine determinate, pentru planificarea ModX-urilor în cicluri cu număr constant de execuții. Ţinând cont şi de caracteristicile de comportare temporală a planificatorului online, au fost concepute şi demonstrate condițiile de necesitate ale planificabilității sistemului (Teorema 5-1). Algoritmul de planificare non-preemptivă pentru astfel de sisteme de task-uri a fost prezentat şi verificat prin simulări şi exemple.

Am introdus şi abordat problema modelării şi planificării seturilor de task-uri ce necesită o execuție fixă în cadrul fiecărei perioade a acestora. A fost introdusă noțiunea de ModX cu execuție fixă (FModX) şi un model matematic al acestuia, bazat pe artimetica modulară şi pe funcția treaptă unitate, a fost conceput (Definiția 5-4) şi demonstrat. În continuare, am studiat planificarea non-preemptivă a seturilor compuse din două FModXuri independente, pe baza funcției de mapare (Definiția 5-5) şi a proprietăților sale, demonstrate în Teorema 5-2 şi Corolarul 5-1. Condițiile de necesitate şi suficiență ale planificabilității au fost introduse şi demonstrate (Teorema 5-3, Corolarul 5-2 şi 5-3), ajungându-se la o formulă simplă, ce specifică faptul că dacă suma duratelor de execuție ale celor două FModX-uri nu depăşeşte valoarea celui mai mare divizor comun al perioadelor lor, atunci setul este fezabil. Algoritmul

de planificare ce utilizează funcția de mapare a fost conceput şi exemplificat la finalul Capitolului 5.

În fine, am prezentat modelul unui sistem complet care unifică toate conceptele introduse şi studiate în lucrare: OPEN-HARTS (Capitolul 6). Sistemul OPEN-HARTS este compus din două subsisteme principale:

- INVERTA – un mediu integrat, vizual, destinat dezvoltării, specificării şi verificării formale, programării şi analizei aplicațiilor TR stricte;
- HARETICK – un nucleu de operare TR hibrid, multi-tasking şi single-user, pentru sisteme de control încorporat (*embedded systems*), conceput pentru a oferi predictibilitate maximă de execuție aplicațiilor TR critice.

Sistemul OPEN-HARTS este conceput ca un set de soluții pentru problemele ridicate de dezvoltarea unitară a unui sistem sau aplicații TR stricte, utilizând modelele de timp sistem, de semnale şi ModX-urile cu planificare şi execuție non-preemptivă, introduse în lucrare:

- Mediul integrat INVERTA permite efectuarea, în regim offline, a întregului set de operații implicate în dezvoltarea şi analiza unei aplicații TR stricte:
  - 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 cu strictețe pe durata operării şi 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 în care e vorba despre un ModX;
  - Programarea aplicației este realizată pentru fiecare task în parte (speficificarea funcțională), utilizând un limbaj de programare (asamblare, C, etc.) modificat corespunzător (restricții de limitare a structurilor şi secvențelor, în timp şi spațiu, 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.
- Nucleul TR hibrid HARETICK permite executarea aplicațiilor TR dezvoltate cu mediul INVERTA, oferind două contexte de execuție: contextul non-preemptiv este destinat execuției cu predictibilitate maximă a task-urilor TR stricte ale aplicației (ModX-urile), garantând respectarea termenelor

specificate, iar contextul preemptiv este destinat execuției task-urilor TR lejere, fără a considera timpul ca o coordonată a operării acestora.
- Încărcarea şi executarea unei noi aplicații pe o platformă țintă pe care rulează HARETICK se face prin înlocuirea celei care se execută în mod curent pe platforma țintă, 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 soluție pentru problemele de predictibilitate a analizei planificabilității non-preemptive în cazul abordărilor online, şi, prin definirea nucelului TR hibrid, se soluționează (cel puțin parțial) problema eficienței scăzute a execuției non-preemptive.

7.2 Rezumat al contribuțiilor

Referatul de față avansează următoarele idei principale:
- Utilizarea întreruperilor în sistemele TR stricte reprezintă o problemă din punctul de vedere al predictibilității, afectând capacitatea acestora de a garanta în orice condiții de operare respectarea termenelor impuse task-urilor TR stricte;
- Pentru a oferi predictibilitate maximă sistemelor TR stricte destinate aplicațiilor critice de achiziție 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 aplicațiilor TR stricte trebuie să aibă la bază, pentru toate fazele sale, metode şi modele omogene, compatibile.

Având la bază aceste principii, lucrarea aduce următoarele contribuții în domeniul sistemelor TR stricte pentru aplicații critice:
- definirea unui model temporal ce permite specificarea comportamentului în timp al semnalelor şi task-urilor unui STR strict;
- 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 temporal 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;

- evidențierea şi exemplificarea avantajelor şi dezavantajelor planificării şi execuției non-preemptive a task-urilor TR;
- evidențierea necesității introducerii etapei de analiză offline a planificabilității setului de task-uri, cu confirmarea (sau nu) a fezabilității 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 optimalității lui *EDFNP*;
- abordarea problemelor de evaluare a fezabilității setului de task-uri utilizând condiții de necesitate şi/sau suficiență;
- demonstrarea faptului că pentru modelul de task-uri considerat (ModX-uri independente, cu perioadele aliniate la momentul inițial $t_0 = 0$), condiția a doua de necesitate enunțată în [Jeffay 91] nu este aplicabilă;
- discutarea cazurilor, mai apropiate de realitate, ale sistemelor TR stricte ce execută, pe lângă setul de ModX-uri ale aplicației, şi un task special – planificatorul online, ce utilizează o tabelă de dispatch de dimensiuni fixe, bine determinate, pentru planificarea ModX-urilor în cicluri cu număr constant de execuții;
- ținând cont şi de caracteristicile de comportare temporală a planificatorului online, au fost concepute şi demonstrate condițiile de necesitate ale planificabilității sistemului;
- prezentarea şi verificarea prin simulări şi exemple a algoritmului de planificare non-preemptivă pentru astfel de sisteme de task-uri;
- abordarea problemei modelării şi planificării seturilor de task-uri ce necesită o execuție fixă în cadrul fiecărei perioade a acestora;
- introducerea noțiunii de ModX cu execuție fixă (FModX) şi demonstrarea unui model matematic al acestuia, bazat pe artimetica modulară şi pe funcția treaptă unitate;
- studierea planificării non-preemptive a seturilor compuse din două FModXuri independente, pe baza funcției de mapare şi a proprietăților demonstrate ale acesteia;
- introducerea şi demonstrarea condițiilor de necesitate şi suficiență ale planificabilității non-preemptive a seturilor de două FModX-uri, ajungându-se la o formulă simplă, ce specifică faptul că dacă suma duratelor de execuție ale celor două FModX-uri nu depăşeşte valoarea celui mai mare divizor comun al perioadelor lor, atunci setul este fezabil;
- algoritmul de planificare ce utilizează funcția de mapare a fost conceput şi exemplificat pentru seturi de câte două FModX-uri;
- 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 aplicațiilor 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 execuție aplicațiilor TR critice);

- evidențierea faptului că prin conceptele introduse de către OPEN-HARTS se oferă o soluție pentru problemele de predictibilitate a analizei planificabilității non-preemptive în cazul abordărilor online, şi, prin definirea nucelului TR hibrid, se soluționează (cel puțin parțial) problema eficienței scăzute a execuției non-preemptive.

7.3 Perspective de cercetare şi dezvoltare

Preocupările curente şi de viitor imediat din cadrul activității noastre de doctorat cuprind următoarele subiecte ce necestită încă a fi rezolvate:

- Verificarea practică a seturilor de formule stabilite între parametrii temporali ai semnalelor şi task-urilor aplicațiilor TR, pe baza modelului temporal propus. Aceste formule constituie fundamentul fazei de verificare şi validare formală a aplicațiilor TR. Evaluarea necesității de formalizare a întregului sistem de modele, utilizând algebre şi logici temporale.
- Enunțarea şi demonstrarea teoremei generalizate de tratare a semnalelor de intrare cu task-uri periodice.
- Abordarea planificării seturilor compuse dintr-un număr oarecare de ModX-uri cu execuție fixă.
- Planificarea seturilor de ModX-uri cu dependențe de program (cu alte cuvinte, planificarea unei aplicații TR complete).
- Integrarea tuturor cazurilor de planificare non-preemptivă şi conceperea unui model unificat, capabil a fi utilizat în cât mai multe aplicații reale.
- Rafinarea specificațiilor, implementarea şi testarea mediului integrat de dezvoltare a aplicațiilor TR (INVERTA): definirea unui limbaj adecvat de programare destinat descrierii funcționale a task-urilor aplicației (este vizat un subset al limbajului C, care să permită însă doar construcții limitate în ceea ce privește durata de execuție 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 aplicației.
- Proiectarea şi realizarea unui analizor temporal al programelor aplicației TR, care să poată extrage informații cu privire la durata maximă de execuție (WCET). Integrarea acestuia în mediul INVERTA.
- Integrarea algoritmului unificat de planificare non-preemptivă a task-urilor TR stricte în mediul de dezvoltare a aplicațiilor TR, pentru a se implementa faza de analiză (offline) a planificabilității aplicației.

În cel de al treilea referat se va prezenta în detaliu un studiu de caz privind proiectarea şi implementarea nucleului de operare RT hibrid HARETICK pe un sistem DSP.

Referințe bibliografice

[Acho 86] A. A. V. Acho, R. Sethi, J. D. Ullman, "Compilers: Principles, Techniques, Tools", Addison Wesley, Reading, Massachusetts, 1986.

[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.

[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.

[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).

[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.

[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).

[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).

[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.

[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).

[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.

[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).

[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.

[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).

[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).

[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.

[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).

[Muchnick 97] S. S. Muchnick, "Advanced Compiler Design and Implementation", Academic Press, Morgan Kaufmann Publishers, 1997.

[Niehaus 91] D. Niehaus, "Program Representation and Translation for Predictable Real-time Systems", Department of Computer and Information Science, University of Massachusetts, 1991.

[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.

[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).

[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).

[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.

[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).

[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).

[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).

[Stankovic 88] J. A. Stankovic, "Misconceptions About Real-time Computing", IEEE Computer, Vol. 21, No. 10, October 1988, pp. (10-19).

[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.

[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).

[Young 82] S. Young, "Real Time Languages: Design and Development", Ellis Horwood Publishers, 1982.