Timişoara

ă şi Calculatoare

e

CONTRIBUȚII LA PROIECTAREA ŞI DEZVOLTAREA APLICAȚIILOR TIMP REAL UTILIZÎND PLC-uri

Teză de doctorat

Autor:

ing. Zmaranda Doina

Coordonator ştiinţific:

prof. dr. ing. Crețu Vladimir

Timişoara
2001

CUPRINS

1. INTRODUCERE

1.1. Oportunitatea tezei

Sistemele timp real reprezintă o categorie largă de sisteme, în prezent din ce în ce mai des întîlnite în practică. Operarea în timp real se deosebeşte în mod fundamental de celelalte forme de procesare prin implicarea explicită a noţiunii de timp. Caracteristicile de bază ale acestui mod de operare pot fi sintetizate de următoarea definiţie, exprimată de standardul DIN 44 300 al industriei germane:

Definiţie:
Modul de operare al unui sistem de calcul în cadrul căruia programele pentru procesarea datelor provenite din exterior operează continuu, astfel încît rezultatele acestora vor fi disponibile în cadrul unor perioade predeterminate de timp.

Definiţia de mai sus implică nu atît viteză de procesare, cît obţinerea unor reacţii într-un timp predefinit, determinat. Această cerinţă a sistemelor timp real face ca dezvoltarea unor astfel de sisteme să fie mult mai complicată ca şi realizare decît în cazul sistemelor care nu operează în timp real.

In trecut, programarea aplicaţiilor timp real s-a bazat pe utilizarea extensivă a limbajelor de asamblare, în primul rînd din considerente de viteză, avînd însă în rest toate dezavantajele cunoscute ale acestora [WM85a,b]. Programarea la nivel de asamblare este însă destul de greoaie, iar pentru sistemele complexe, sursa rezultată este foarte mare, fiind extrem de greu de depanat şi de întreţinut. In prezent insă, utilizarea asamblării este din ce în ce mai rară din cauza apartiţiei unor unelte software care permit dezvoltarea integrată a aplicaţiilor timp real, începînd cu configurarea hardware şi terminînd cu proiectarea şi implementarea software.

In ciuda acestor unelte, actualmente domeniul dezvoltării aplicaţiilor timp real se află încă într-un stadiul de pionierat, în sensul că majoritatea sistemelor de acest tip au fost şi sunt dezvoltate utilizînd tehnici ad-hoc, fără a avea o bază ştiinţifică [Gom86]. Aceste metode au fost dezvoltate pe baza necesităţii de a rezolva rapid anumite probleme apărute în practică, dar soluţiile adoptate au de cele mai multe ori un caracter specific, non-general. Chiar dacă nu furnizează toate facilitaţile necesare, aplicarea pe scară largă a acestor unelte CASE existente în prezent poate uşura considerabil efortul de dezvoltare a unei aplicaţii timp real complexe. Este cu totul greşită ideea că apariţia unor procesoare şi sisteme de comunicare din ce în ce mai performante va rezolva problemele legate de aspectele temporale în sistemele timp real.

Dimpotrivă, va exista tendința de a crea sisteme concurente tot mai complexe, a căror probleme legate de cerințele temporale vor creşte în loc să scadă [Sta88]. Bineînțeles că performanțele tot mai crescînde vor putea fi exploatate şi în acest caz, dar aceasta nu înseamnă că cerințele temporale vor fi îndeplinite în mod automat. Fără o ***proiectare adecvată*** a aplicației date, conform unei metodologii bine stabilite nu este sigur că procesoarele împreună cu sistemul de comunicatie vor putea face față tuturor situațiilor de încărcare maximă care pot apărea în cadrul unei aplicații concrete [KS97].

Există la ora actuală o serie de medodologii de dezvoltare software larg utilizate pentru aplicațiile care nu au componentă de timp real; acestea sunt cunoscute din domeniul ingineriei programării dar ***nu sunt aplicabile decît eventual parțial pentru sistemele timp real***. Acest lucru se datorează în principal faptului că, în cazul aplicațiilor timp real există alte criterii de optimalitate şi performanță care trebuiesc luate în considerare.

Astfel, de exemplu, pentru sistemele software clasice, utilizarea la maximum a procesorului reprezintă un criteriu esențial de proiectare de care se ține cont în majoritatea metodologiilor existente şi care a făcut subiectul a numeroase articole în literatura de specialitate. Pentru sistemele timp real însă, este complet irelevant faptul că utilizarea procesorului nu este optimă; utilizarea slabă a procesorului este un preț mic care trebuie plătit în cazul în care este obținută în schimb o mai mare simplitate a software-ului ca şi o premiză de bază în obținerea dependabilității şi predictabilității ulterioare a sistemului.

De asemenea, in cazul sistemelor distribuite clasice, sunt binecunoscute schemele de încarcare a nodurilor cu task-uri şi modalitățile de migrare a task-urilor de la un nod la altul în cazul apariției situațiilor de încărcare maximă. In sistemele timp real însă, astfel de idei sunt în general ne-aplicabile, deoarece ele pot să se refere doar la task-urile de procesare propriu-zisă. In contrast cu acestea, task-urile de control a diferitelor mărimi din sistem sunt însă puternic limitate de I/O; legarea permanentă a unor dispozitive externe (senzori, actuatori) de anumite noduri face practic ca o astfel de politică de ***load-sharing*** să fie imposibil de utilizat. In consecință, cercetarea în acest sens trebuie să fie orientată funcție de aceste realități [Hal92].

In general, gîndirea în termeni probabilistici sau statistici, lucru comun în ştiința calculatoarelor în ceea ce priveşte evaluarea performanțelor, este nepotrivită în cazul domeniului timpului real; la fel este şi stabilirea criteriului de minimizare a timpului mediu de răspuns ca şi criteriu de optimalitate utilizat în procesul de proiectare. In locul acestora, trebuiesc luate în considerare în procesul de proiectare criterii ca de exemplu următoarele [Kop97]:

- cazurile cele mai defavorabile
- timpii limită (deadline)
- întîrzierile maxime posibile, etc.

Pentru a realiza sisteme timp real ***dependabile*** şi ***predictibile***, este necesară o proiectare bazată pe o gîndire în termeni statici, precum şi pe acceptarea tuturor constrîngerilor fizice existente; toate elementele dinamice sau virtuale care ar putea fi utilizate sunt considerate în acest caz dăunătoare [AMR98].

In mod evident, un grad înalt de dependabilitate nu înseamnă în mod necesar că sistemul nu se va defecta niciodată; nici un sistem tehnic nu este fiabil la modul absolut [McD93]. Totuşi, utilizînd măsurile corespunzătoare încă din faza de proiectare se poate face ca violarea unor deadline-uri precum şi consecinţele acestora să fie cuantificabile şi cît mai puţin probabile. Astfel, limitările individuale ale unui sistem timp real trebuie recunoscute şi riscurile utilizării acestuia trebuie avute în considerare.

In concluzie, în procesul de proiectare a aplicaţiilor timp real efortul trebuie dirijat către ***definirea unor metodologii care să asigure în primul rînd cerinţele specifice unor astfel de sisteme***, şi anume: îndeplinirea cerinţelor temporale, predictabilitate şi dependabilitate. Aceste obiective pot fi realizate în primul rînd prin alegerea **simplităţii** ca principiu fundamental al proiectării, conform următorului lanţ cauzal:

Simplitate => Predictabilitate => Dependabilitate

Astfel, definirea unei metodologii pornind de la principiul de mai sus reprezintă o ***necesitate*** în practica actuală de dezvoltare a sistemelor timp real şi care rezolvă pentru moment o largă categorie de aplicaţii critice. Lucrarea de faţă propune o astfel de metodologie. In pofida simplităţii şi deci şi a limitărilor inerente, metodologia propusă poate constitui o bază de pornire în dezvoltarea ulterioară a altor metodologii, mult mai complicate, bazate eventual şi pe utilizarea metodelor formale. Pentru practica actuală însă, datorită imaturităţii metodelor formale în ceea ce priveşte sistemele timp real complexe, metodologia propusă reprezintă o unealtă extrem de utilă în dezvoltarea sistematică în special a aplicaţiilor de control timp real, ca o premiză esenţială în realizarea deziderateloor de predictabilitate şi determinism.

1.2. Sisteme software. Sisteme timp real

Activitatea de dezvoltare a software-ului este, de cele mai multe ori, doar o componentă din activitatea de proiectare a unui sistem, mult mai complexă în ansamblul ei, şi în cadrul căreia cerinţele software-ului trebuie să se alinieze la cerinţele sistemului în întregul său. Sistemele de monitorizare şi control, sistemele bancare, etc., sunt numai cîteva exemple în acest sens.

Astfel, pentru o proiectare cît mai corectă, sunt necesare nu numai cunoaşterea problemelor legate de dezvoltarea componentei software propriu-zise şi a interfeţelor abstracte cu care aceasta trebuie să interacţioneze, dar şi informaţii despre modul de funcţionare al sistemului de proiectat în ansamblu. Rezultă deci, că o astfel de activitate este de cele mai multe ori o activitate interdisciplinară şi care necesită, în mod imperativ lucrul în echipă. Utilizarea tehnicilor ingineriei programării este esenţială, în măsura în care acest lucru este posibil din punct de vedere practic.

Un sistem software se dezvoltă în general cu scopul de a automatiza o anumită aplicaţie particulară: din acest motiv se poate caracteriza un sistem software funcţie de cerinţele aplicaţiei propriu-zise. Astfel, există patru mari tipuri de sisteme software, şi anume [GJM91]:

- sisteme informaţionale
- sisteme timp real
- sisteme distribuite
- sisteme încorporate

Clasificarea de mai sus are mai mult un caracter orientativ pentru că în realitate multe sisteme prezintă caracteristici comune mai multor tipuri dintre cele prezentate mai sus. Astfel, putem avea sisteme informaţionale cu cerinţe de timp real, eventual şi distribuite; mai mult, un astfel de sistem poate fi încorporat într-un sistem mai mare de monitorizare şi control. De altfel, sistemele încorporate sunt deseori, prin natura lor, sisteme cu cerinţe de timp real.

Definiţie:
Un sistem timp real este un sistem în cadrul căruia funcţionarea corectă nu depinde numai de rezultatele logice ale sale dar şi de momentul în care acestea au fost obţinute.

Cu alte cuvinte, un sistem în timp real este orice sistem în care ***momentul*** în care rezultatele sunt obţinute este semnificativ. Dinamica domeniului a făcut însă de multe ori ca sfera noţiunii de timp real să devină instabilă şi vagă. Definiţiile prea tranşante, încorsetate în terminologii riscă să devină neacoperitoare pentru o largă categorie de aplicaţii care există în prezent [BF97].

De asemenea, de multe ori noțiunea de timp real s-a restrîns, în special la domeniul controlului proceselor tehnologice sau conducerii operative a producției, din motive istorice.

Indiferent de formulare, definiția unui sistem timp real cere ca rezultatele să fie obținute în timp util pentru a putea avea efect nemijlocit asupra sistemului şi nu doar o valoare pur informativă, ca în cazul tradițional. De asemenea, se poate observa că definiția este foarte generală, atotcuprinzătoare, acoperind practic orice tip de sistem care respectă cerințele de timp. De asemenea, tot din definiție rezultă că nu este obligatoriu ca sistemul să intreprindă el însuşi ceva, este suficient să furnizeze rezultatele care să permită o eventuală intervenție din afară (în speță a omului); deci, teoretic, supravegherea este suficientă pentru calitatea de timp real.

În fine, trebuie menționat că viteza de reacție nu este precizată în valori absolute (adică nu se impune nici un timp de răspuns, nici măcar ***foarte repede***), ci relativ, în raport cu viteza de desfăşurare a proceselor în cauză şi cu implicațiile acestora. Astfel, de exemplu, cîteva ms pot însemna prea încet pentru supravegherea unui reactor nuclear, iar o jumătate de oră poate fi suficient de repede pentru cazanul unei centrale termice. Reciproc, o viteză fulger este total inutilă dacă rezultatele prelucrării se aplică mai încet decît permite contextul.

Figura 1 descrie structura generală a unui sistem timp real [Kop97] :

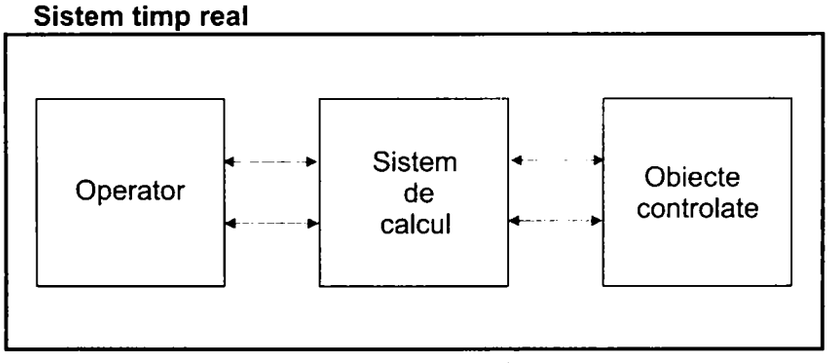


Figura 1: Structura generală a unui sistem timp real

După cum se poate observa, o caracteristică foarte importantă a unui sistem timp real o reprezintă faptul că acesta interacționează cu o mare varietate de obiecte externe calculatorului pe care le controlează. Interfața cu obiectele externe constă în principal din actuatori şi senzori care transformă semnalele fizice (tensiuni, curenți, debite, temperaturi, etc.) într-o formă digitală pentru a putea fi interpretate de calculator; interfața om-maşină constă în principal din dispozitive pentru introducere/extragere de date (tastatură respectiv monitor).

Practic, cele mai stringente cerințe temporale a sistemelor de timp real îşi au de fapt originea în cerințele buclelor de reglare a mărimilor şi/sau obiectelor controlate.

O altă caracteristică importantă a sistemelor timp real o reprezintă faptul că acestea trebuie să fie uşor de întreținut şi actualizat: trebuie să existe posibilitatea de reconfigurare ulterioară a acestuia astfel încît schimbările efectuate să cauzeze cît mai puține perturbări în sistem. Aceasta deoarece majoritatea sistemelor timp real sunt destinate să opereze continuu ani de zile, şi deci în procesul de proiectare al acestora nu trebuie să se presupună că acestea operează într-un mediu static.

O categorie aparte de sisteme timp real o reprezintă ***sistemele utilizate pentru conducerea şi controlul proceselor industriale***, acestea constînd în principal dintr-un proces fizic şi un sistem de control format la rîndul său atît dintr-o componentă hardware cît şi dintr-o componentă software. În cele mai multe cazuri, sistemele de control timp real sunt sisteme încorporate: adică calculatorul este doar un element funcțional din cadrul sistemului în ansamblul său. Astfel, o caracteristică comună a sistemelor înglobate este aceea că, calculatoarele sunt conectate la mediul în care lucrează printr-o mare varietate de dispozitive de interfațare şi recepționează şi transmit o mare varietate de stimuli [Ben94].

De exemplu, intrările şi ieşirile unui sistem (I/O) precum şi task-urile privind comunicarea au o caracteristică comună: sunt conectate prin intermediul unor dispozitive către procese care sunt exterioare calculatorului. Aceste procese operează în propria lor scară de timp, iar despre calculator se spune că operează în timp real dacă acestea se aliniază cu viteza de desfăşurare a proceselor.

Cu cît astfel de sisteme sunt mai complexe, cu atît mai mult specificarea, proiectarea şi implementarea acestora devine mai dificilă. In prezent, complexitatea aplicațiilor de acest gen este într-o continuă creştere datorită a numeroşi factori, printre care:

- creşterea complexității sistemelor şi proceselor controlate
- creşterea cerințelor de de siguranță şi fiabilitate
- necesitatea utilizării de sisteme distribuite, cu tot ceea ce implică acestea din punctul de vedere al comunicării între diferitele tipuri de subsisteme, etc.

Datorită acestui fapt, noi modele şi metode sunt necesare a fi utilizate în cadrul ciclului de viață a unor astfel de aplicații. Astfel, modelele dezvoltate în acest scop pot fi ***descriptive*** (informale sau semi-formale) sau ***prescriptive*** (formale):

- ***modelele descriptive*** pot fi informale sau semiformale, in cadrul lor se utilizează notații grafice sau textuale; o caracteristică generală a acestora este aceea că ele sunt simple însă suficient de exacte pentru a conține toate informațiile legate de sistemul care urmează a fi proiectat
- ***modelele prescriptive*** diferă de cele descriptive prezentate prin aceea că sunt formale; din acestă cauză, verificarea şi validarea acestora este posibilă şi, de asemenea, transpunerea acestora într-un limbaj de programare poate fi realizată fără ambiguități, în anumite cazuri chiar automatizată.

In consecință, pentru dezvoltarea unor astfel de sisteme complexe, este necesară utilizarea combinată atît a tehnicilor din cadrul domeniului de ***ingineria controlului automat*** **ICA** cît şi din cadrul ***ingineriei programării*** **IP.**

1.3. Ciclul de viață al software-ului şi sistemele de control timp real

Proiectarea şi implementarea sistemelor timp real urmează aceiaşi paşi cunoscuți de la ingineria programării privind ciclul de viață al software-ului, ca şi orice alt tip de sistem software. Totuşi, avînd în vedere caracteristicile particulare ale acestor sisteme, sunt necesare unele precizări referitoare la aspectele specifice acestora.

1.3.1. Modele din domeniul ingineriei controlului automat (ICA)

Clasificările modelelor utilizate în domeniul controlului automat sunt în principal bazate pe natura sistemului modelat, de exemplu, sisteme liniare versus sisteme ne-liniare. Acest lucru se datorează faptului că majoritatea metodelor utilizate pentru analiza şi proiectarea sistemelor liniare nu pot fi aplicate în cazul sistemelor non-liniare şi vice-versa. Un alt exemplu de clasificare în contextul modelelor utilizate pentru sistemele de control se referă la modele continue versus modele discrete [Per91].

Clasificarea modelelor poate porni însă şi de la modul în care a fost construit modelul însuşi: atunci cînd este posibilă derivarea acestuia pe baza unor legi fizice cunoscute se obține un model de tipul **white box**. Alternativa o constituie un model derivat pe baza unor date măsurate, aşa numitul model **black box** [FB95].

Indiferent însă de clasificarea utilizată, caracteristica comună a modelelor din cadrul domeniului **ICA** este aceea că ele sunt modele matematice. Acest lucru

este firesc, avînd vedere faptul că acestea sunt utilizate pentru proiectarea algoritmilor de control. Astfel, dezvoltarea unui sistem de control automat poate fi descrisă de următoarele faze [Boa93], prezentate în cadrul Figurii 2.

- ***analiza şi modelarea procesului*** - obiectivul acestei faze este acela de a modela procesul de controlat; de asemenea, de a identifica principalii parametri ai acestuia precum şi a ipotezelor care ar putea conduce la o simplificare a modelului generat. Validarea acestui model se face de regulă utilizînd simularea
- ***analiza şi proiectarea controlului*** – în cadrul acestei faze este formulată problema de control şi sunt proiectați algoritmii de control în consecință. Validarea strategiei de control adoptate în cadrul acestei faze este realizată de asemenea prin simulareîn dar în strînsă legătură cu modelul procesului de controlat, care a fost realizat în cadrul fazei anterioare
- ***faza de implementare*** – în cadrul acestei faze este realizată implementarea propriu-zisă a algoritmilor de control şi testarea acestora

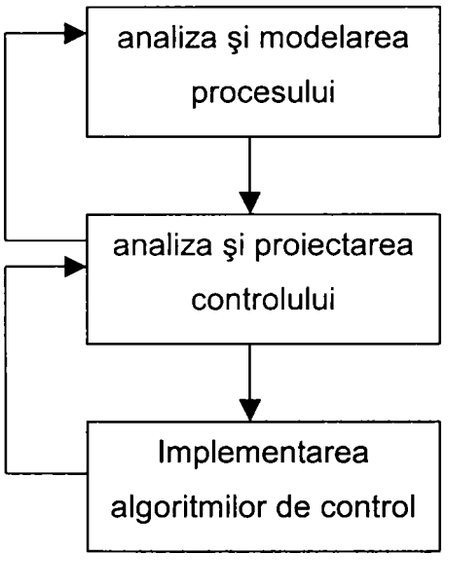


Figura 2: Fazele de dezvoltare al unui sistem de control automat

Se observă că procesul de dezvoltare nu este strict secvențial, existînd un **feed-back** între faza de modelare a procesului şi cea de proiectare a controlului. De asemenea, se observă că implementarea şi testarea propriu-zisă a codului este realizată într-o singură fază, fapt care este relativ dificil de realizat în cazul sistemelor complexe. Acest lucru ridică problema posibilității utilizării uneltelor şi tehnicilor din cadrul ingineriei programării **IP** în cadrul acestei faze de dezvoltare a sistemelor de control automat.

1.3.2. Modele din domeniul ingineriei programării (IP)

În general, de la începutul său şi pînă la furnizarea către client, un sistem software se află într-o continuă dezvoltare şi modificare. Aceasta se numeşte ciclul de viaţă al software-ului şi, după cum este bine cunoscut din cadrul ingineriei programării **IP**, se compune din mai multe faze, fiecare din aceste faze rezultînd în dezvoltarea fie a unei părţi a sistemului, fie a ceva asociat acestuia (cum ar fi de exemplu o documentaţie) [Pfl91][GJM91].

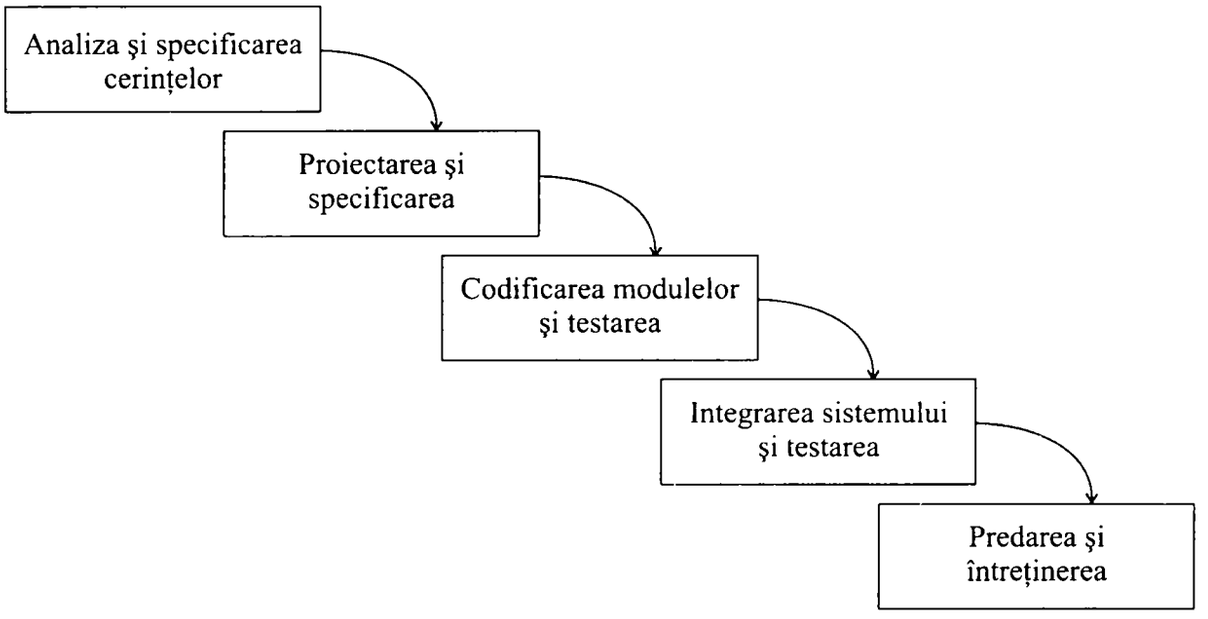


Figura 3: Modelul *cascadă* al ciclului de viaţă al software-ului

În cadrul modelului teoretic tradiţional, aşa numitul ***model cascadă*** (Figura 3), fiecare fază are bine delimitate punctele de început şi sfîrşit, iar interfeţele cu fazele ulterioare sunt clar definite [GJM91]. Modelul general cascadă include următoarele etape:

- ***studiul de fezabilitate, specificarea şi analiza cerinţelor*** - obiectivul acestei faze este acela de a specifica funcţiile care sunt cerute sistemului de dezvoltat. Interfeţele cu caracter general cît şi cerinţe de performanţă sunt specificate în detaliu. Rezultatul acestei faze trebuie validat, din punctul de vedere al satisfacerii cerinţelor
- ***proiectarea şi specificarea sistemului***, care include atît ***proiectarea arhitecturala*** (sau de ansamblu) cît şi ***proiectarea detaliată***; în cadrul acestei faze se defineşte arhitectura hardware şi software a sistemului. În partea finală a acestei faze proiectarea detaliată va cuprinde specificarea completă a structurilor de date şi control, a parametrilor de intrare/ieşire şi a algoritmilor utilizaţi

- ***codificarea modulelor şi testarea*** - fiecare modul este codificat şi testat separat
- ***integrarea sistemului şi testarea*** - în cadrul acestei faze modulele sunt integrate în sistem şi se verifică sistemul apoi în ansamblul său
- ***predarea sistemului şi întreținerea***

Acest model teoretic, foarte simplu, nu poate fi însă aplicat în practică sub această formă, mai ales pentru sisteme timp real, din diferite motive. De exemplu, dacă testele descoperă erori majore în sistem şi care nu pot fi corectate pe loc, trebuie revenit fie în faza de codificare, fie, în funcție de profunzimea erorilor, chiar în faza de proiectare. De asemenea, dacă în faza de proiectare sunt descoperite inconsistențe sau ambiguități, trebuie revenit chiar în faza de specificare a cerințelor. Deci, în general, o eroare dintr-o anumită fază necesită o revenire cel puțin într-o fază anterioară, dacă nu mai mult.

O altă simplificare a modelului este aceea că, se presupune că o anumită fază este terminată în momentul în care se trece la următoarea, lucru care în practică, datorită ne-disponibilității totale a unor date, nu este întotdeauna posibil. Astfel, pentru a permite revenirea într-o fază anterioară într-o manieră disciplinată, modelul a fost completat cu bucle de feed-back către fazele precedente, în scopul de a minimiza pe cît posibil necesarul de muncă atunci cînd este cazul să se revină (Figura 4), rezultînd aşa numitul model ***cascadă cu revenire*** [GJM91]:

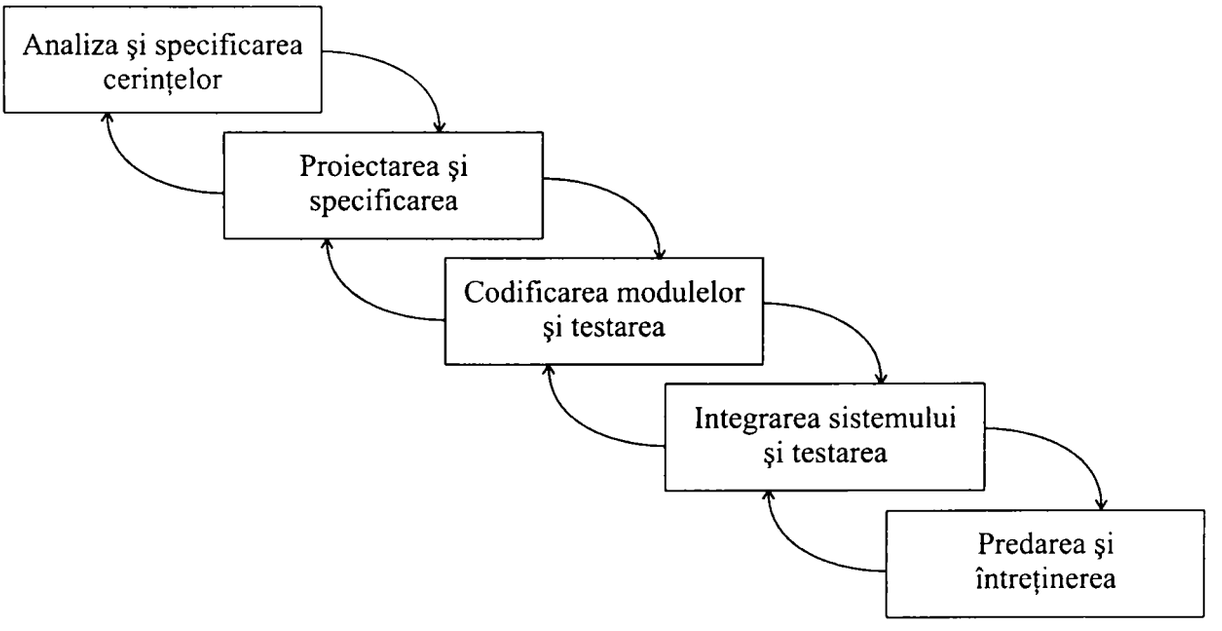


Figura 4: Modelul "cascadă" cu revenire

Definirea unor modele, ca cele de mai sus, cu scopul de a organiza în mod eficient construcția sistemelor software este deosebit de utilă, şi se bazează pe ideea că produsele software, asemenea celor industriale, au un ciclu de viață care se întinde de la concepția inițială şi pînă la retragerea sa, şi că acest ciclu de viață trebuie anticipat şi controlat în vederea obținerii calității dorite. Totuşi, avînd în vedere specificul produselor software, două caracteristici importante le disting pe acestea de alte tipuri de produse:

- producția software este prin excelență o activitate intelectuală, şi deci, nu este uşor automatizabilă
- software-ul se caracterizează printr-o mai mare instabilitate: cerințele se schimbă constant şi ca o consecință, produsul trebuie să poată evolua uşor

În concluzie, modelele tip ***cascadă***, în toate variantele acestora, au meritul de a introduce mai multă disciplină în procesul de dezvoltare software, dar această disciplină este însoțită de oarecare rigiditate. Această rigiditate poate crea probleme, mai ales în cazul în care software-ul este dezvoltat pe baza unor cerințe puțin sau greşit înțelese, cum adesea se întîmplă. De asemenea, mai ales în cazul sistemelor în timp real, software-ul trebuie să evolueze continuu, funcție de schimbările intervenite în sistemul de care face parte, constituind astfel o entitate dinamică. Constrîngerea de a dezvolta un astfel de software într-o manieră rigidă, monolitică, aşa cum impune modelul cascadă, este nenaturală şi neproductivă: evoluția în acest caz nu poate fi separată de dezvoltare ci trebuie privită ca parte componentă a acesteia.

Aceste deficiențe ale modelului cascadă au condus la existența unor alte tipuri de modele, mai mult sau mai puțin aplicabile în prezent în practică. Cel mai des utilizat la ora actuală, modelul ***evolutiv***, se bazează pe ideea unei strategii incrementale: conform acestui model, prima versiune a produsului este văzută ca o încercare a cărui scop principal este acela de a confirma fezabilitatea produsului şi de a verifica cerințele. Apoi această versiune se ignoră, şi procesul de dezvoltare se reia de la început, de data aceasta avînd un fundament mult mai solid, bazat pe rezultatele implementării inițiale.

Strategia de dezvoltare utilizînd modelul evolutiv, poate fi descrisă prin următorii paşi:

- se furnizează o primă versiune utilizatorului
- se evaluează în toate dimensiunile versiunea dată
- se modifică versiunea inițială, atît ca proiectare cît şi ca implementare funcție de feed-back-ul obținut, rezultînd versiunea finală

Practic, prima versiune este un aşa numit ***prototip*** al aplicației, care este utilizat doar temporar, pînă produce suficient feed-back pentru versiunea ulterioară. A doua versiune este apoi dezvoltată utilizînd de această dată modelul cascadă. Principala problemă pe care modelul evolutiv nu o rezolvă o reprezintă anticiparea schimbării.

Un alt model, modelul ***transformare***, rezolvă această problemă, însă în acest moment acesta nu reprezintă o paradigmă aplicabilă practic în producția de software deoarece nu există decît medii experimentale pentru suportul acesteia. Cercetările în acest sens relevă că modelul transformare îşi are rădăcinile în domeniul specificațiilor formale: ideea este că procesul de dezvoltare software poate fi văzut ca o secvență de paşi care transformă într-o manieră incrementală specificațiile în implementare:

- cerințele informale sunt analizate şi funcțiile sunt specificate într-un mod formal, într-o manieră incrementală
- descrierea formală devine executabilă pe un procesor abstract

Descrierea formală poate fi văzută ca un prototip al sistemului: orice transformări ulterioare pot fi realizate prin transformarea începînd cu punctul corespunzător din specificații.

Avantajele utilizării modelelor bazate pe metode formale sunt evidente [Hal90]. În primul rînd, utilizarea unui model formal conduce la o mare precizie şi conciziune în notare. De asemenea, utilizînd o notație formală, aceasta va putea fi analizată şi verificată utilizînd operatori matematici, deci proprietăți ca de exemplu corectitudinea sintactică şi consistența internă pot fi foarte bine testate şi dovedite matematic. Si, nu în ultimul rînd, transformarea într-un limbaj de programare utilizat pentru implementare este foarte simplă, uneori poate fi chiar automatizată, nelăsînd loc la neclarități şi ambiguități.

Dezvoltarea actuală a tehnicilor de analiză formală nu permite încă utilizarea în practică pe scară largă a acestora; în principal, pentru a putea fi aplicabile astfel de tehnici trebuie să rezolve cîteva probleme legate de [HM96]:

- scalabilitate
- simplitate
- furnizarea de unelte corespunzătoare

In plus, față de problemele enumerate mai sus, pentru sistemele timp real, mai trebuie rezolvată şi problema introducerii noțiunii de timp în cadrul tehnicii formale utilizate. Incercări în acest sens există în prezent în două direcții: extinderea metodelor existente şi cunoscute cu facilități de specificare a constrîngerilor de tip şi respectiv dezvoltarea de noi metode formale specifice pentru aplicațiile de timp real şi care să includă de asemenea astfel de facilități.

Dezvoltarea continuă a tehnicilor formale şi a uneltelor aferente dedicate sistemelor în timp real din ultimii ani face ca utilizarea unor astfel de tehnici să devină o cerință importantă pentru sistemele critice în viitorul apropiat. Practic, modelul transformare va fi aplicabil în sistemele actuale numai în momentul în care se va trece de la stadiul de cercetare la nivel de laborator, pentru aplicații de dimensiuni reduse, la extinderea pentru aplicații complexe, cum sunt cele care apar în practică. Problema este însă că, pentru acestea, descrierea formală se complică foarte mult.

1.3.3. Aplicabilitatea modelelor din domeniul ICA şi IP pentru sistemele de control timp real

Pentru sistemele de control timp real trebuie menționat că, deşi ingineria programării a realizat mari progrese în ultimii ani, existînd tehnici standard utilizate în diferite faze ale dezvoltării, totuşi la ora actuală nu s-au conturat încă metode general-acceptate specifice pentru acest domeniu. În practică, de cele mai multe ori, proiectantul realizează un compromis, bazîndu-se pe experiența şi judecata proprie, utilizînd însă, în combinație, şi uneltele pe care are la dispoziție în diferite faze (analiză formală, simulatoare, etc.). Dezvoltarea software-ului pentru controlul timp real al proceselor este de cele mai multe ori înglobată în cadrul dezvoltării globale a întregului sistem, şi care include şi procesul de controlat.

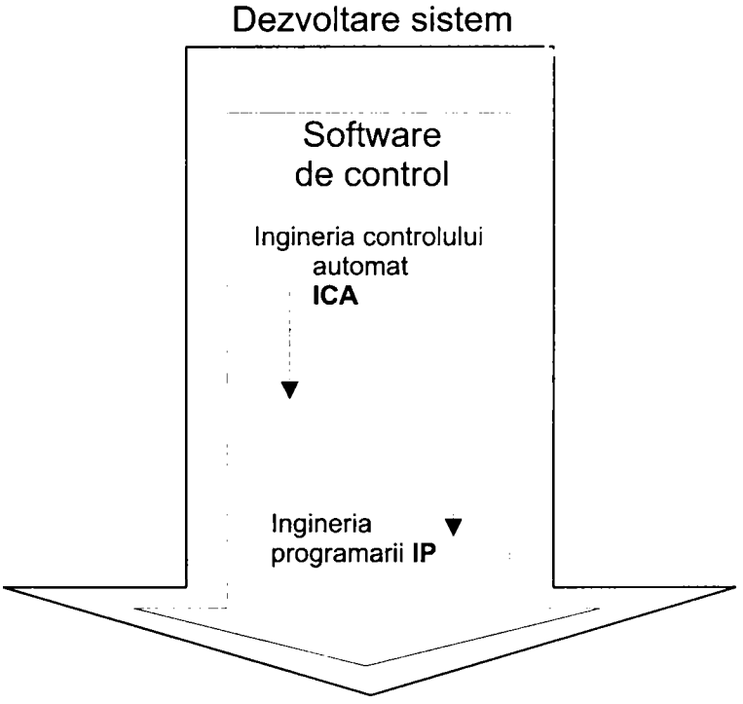


Figura 5: Dezvoltarea software-ului de control înglobată în dezvoltarea sistemului în ansamblul său

Astfel, din punct de vedere al modelului utilizat pentru astfel de sisteme de control timp real, este necesară utilizarea unei combinații între modelele cunoscute de la ingineria programării **IP** şi modelele utilizate în ingineria controlului automat **ICA**. Ilustrarea acestui concept este ilustrat în Figura 5.

În general dezvoltarea integrată a software-ului de control timp real poate fi realizată în trei moduri: secvențial, concurent şi interactiv, conform Figurii 6. In cadrul acesteia, fiecare săgeată ilustrează activitățile legate de unul dintre cele două domenii implicate: ingineria programării **IP** şi respectiv ingineria controlului automat **ICA**. În esență, toate cele trei moduri arată de fapt transformarea unor cerințe din cadrul domeniului **ICA** într-o implementare software specifică domeniului **IP**:

- ***schema de dezvoltare secvențială*** - este caracterizată prin utilizarea succesivă a modelelor din cadrul **ICA** şi apoi a unor modele din cadrul **IP**. Evident, acest lucru este posibil numai dacă transformarea acestor modele dintr-unul într-altul este relativ simplă
- ***schema de dezvoltare concurentă*** - se caracterizează prin utilizarea unui model unic, ca fiind o combinație a modelelor din **IP** şi **ICA**. Un avantaj al acestei soluții îl reprezintă faptul că dezvoltarea este complet integrată şi că, teoretic, este posibil să se determine consecințele deciziilor de proiectare din cadrul domeniului de control automat asupra implementării software-ului propriu-zis şi vice-versa
- o extindere a schemei concurente cu posibilitatea de revenire este prezentată în ***schema de dezvoltare iterativă***. Necesitatea feed-back-ului în procesul de dezvoltare este impusă de practică, pentru a exista posibilitatea detectării specificațiilor incorecte sau incomplete (modelul ***cascadă cu revenire***).

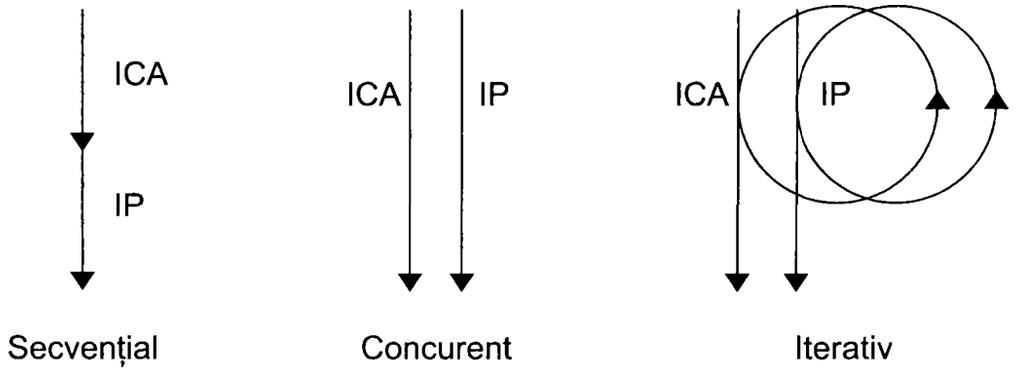


Figura 6: Trei scheme de dezvoltare

1.3.4. Un model de dezvoltare a aplicațiilor de control timp real

Bazat pe aceste considerente, se propune modelul din Figura 7b pentru dezvoltarea sistemelor de control în timp real. Se poate observa din cadrul acestuia că, față de sistemele informatice obişnuite (Figura 7a), modelul de dezvoltare pentru aplicațiile de control timp real prezintă cîteva aspecte şi faze specifice şi care rezultă din combinarea modelelor utilizate în domeniul **ICA** şi **IP**. Practic, în cazul unor astfel de sisteme, dezvoltarea acestora începe cu crearea unui model descriptiv, simplu, al procesului de controlat; acesta trebuie însă să fie suficient de exact pentru a capta toate caracteristicile de bază ale sistemului şi care sunt necesare pentru elaborarea ulterioară a strategiei de control dorite. Modelul este extins ulterior cu strategia de control adoptată, rezultînd un model de data aceasta prescriptiv al sistemului şi care va fi ulterior implementat software.

1.3.4.1. Analiza şi specificarea cerințelor

Etapa de analiză şi specificare (Figura 7b), se ocupă de interpretarea cerințelor utilizatorului în scopul de a produce specificațiile detaliate ale sistemului; aceasta include, aşa cum a fost deja precizat anterior, şi procesul fizic de controlat, precum şi o evaluare a resurselor (în termeni de număr de oameni, timp, echipament, costuri) necesare.

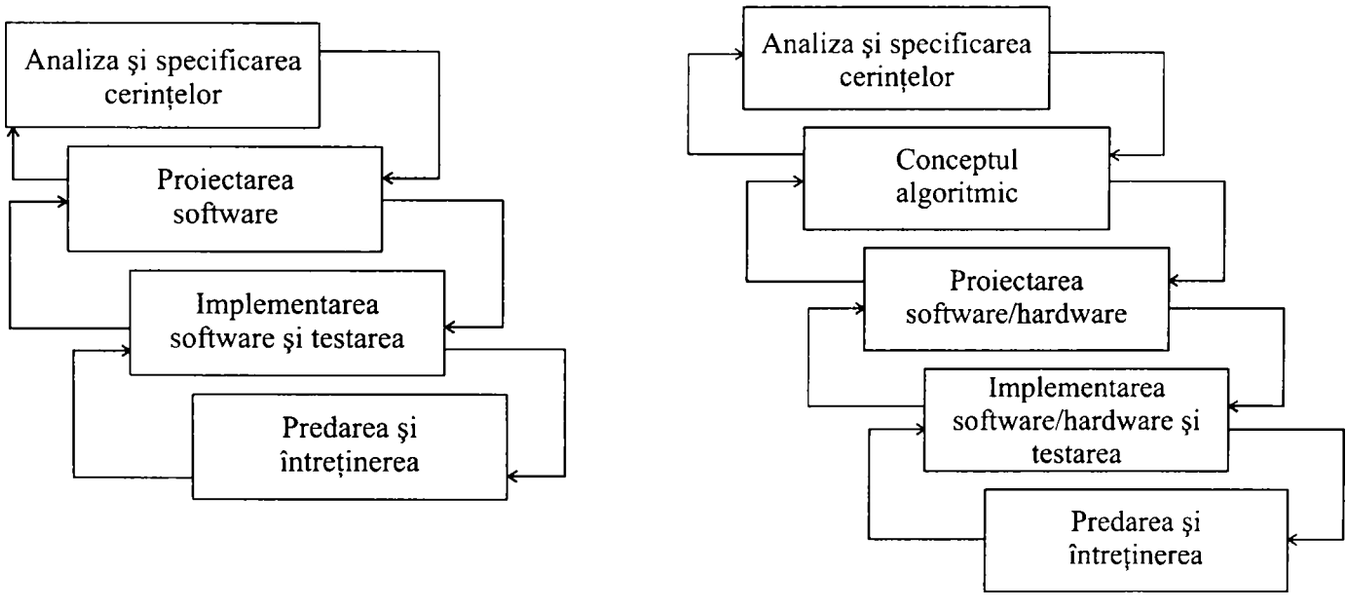


Figura 7: Diferența dintre cilul de viață al sistemelor informaționale (a) şi cele de control timp real (b)

În această fază se poate face şi o apreciere generală asupra structurii de calculatoare dorită: un calculator central, un sistem ierarhic sau un sistem distribuit. Tehnicile generale cunoscute ale **IP** pot fi folosite în cadrul acestei etape în combinație cu tehnicile **ICA** pentru modelarea procesului propriu-zis. Rezultatul acesteia o reprezintă elaborarea unui ***model descriptiv*** P_d al procesului controlat.

1.3.4.2.Conceptul (controlul) algoritmic

Un prim aspect diferit în cazul sistemelor de control timp real față de cele informaționale îl reprezintă necesitatea introducerii, între faza de specificare şi cea de proiectare, a unei faze suplimentare, aşa numitul ***concept algoritmic*** care include strategia de monitorizare şi control adoptată pentru procesul fizic controlat descris în cadrul etapei anterioare. Această fază poate fi considerată şi ca o etapă distinctă a fazei de specificare dar este obligatoriu să fie realizată înaintea începerii proiectării (software şi hardware).

Cu precădere în cadrul acestei etape se vor folosi tehnicile şi modelele din cadrul domeniului **ICA**, care diferă funcție de natura sistemului modelat: de exemplu sisteme liniare versus sisteme non-liniare. Rezultatul acestei etape îl reprezintă definirea unui ***model descriptiv*** C_d pentru strategia de control adoptată. Practic, la o primă trecere, cele două faze inițiale se vor concretiza într-un document care conține specificațiile descriptive ale sistemului în ansamblul său, $M_d = P_d + C_d$. Aşa cum a fost precizat mai sus, în mod special pentru sistemele de control în timp real acest document va conține şi informații legate de conceptul algoritmic utilizat.

Specificațiile informale din cadrul modelului descriptiv M_d vor fi formalizate ulterior, pe măsură ce se revine pe baza feed-back-ului. Astfel, rezultă pe rînd modelul prescriptiv al procesului de controlat P_p, şi apoi modelul prescriptiv al strategiei de control adoptate C_p. In final, prin combinarea acestora va rezulta ***modelul prescriptiv al sistemului***, $M_p = P_p + C_p$.

Se observă utilizarea în cadrul ciclului de viață descris anterior a schemei de dezvoltare iterativă (utilizarea feed-back-ului). Validarea modelului M_d este posibilă deoarece maparea lui M_d pe M_p trebuie să fie combinată cu validarea lui M_p pe M_d pentru a putea realiza feed-back-ul. Verificarea modelului M_p se realizează în domeniul **IP**, în general utilizînd simularea. Se va reveni mai detaliat asupra acestor modele în prezentarea generală a metodologiei de dezvoltare propuse în cadrul capitolului 5.

1.3.4.3. Proiectarea software/hardware

Pe baza modelelor elaborate mai sus, etapa de proiectare se concretizează în definitivarea structurilor globale de date precum şi a arhitecturii software respectiv hardware utilizate. Astfel, din punct de vedere hardware, se vor analiza posibilitățile existente, dintre care, în cazul sistemelor de control în timp real cele mai utilizate sunt următoarele variante [SMY00]:

- un calculator general, central, cu mai multe plăci de achiziție multicanal pentru intrări/ieşiri analogice respectiv digitale
- cîte un calculator de tip general pentru fiecare subunitate funcțională a sistemului
- microcontroller-e separate pentru fiecare subunitate, legate la un calculator general
- un controller programabil cu multiple plăci de I/O legat la un calculator central

Din punct de vedere software, se vor examina specificațiile şi se va decide asupra funcțiilor care trebuiesc realizate. În general, pentru sistemele timp real acestea se încadrează în următoarele categorii:

- prelucrarea intrărilor
- controlul diferitelor mărimi (de exemplu temperaturi, presiuni, debite, etc.)
- actualizarea ieşirilor
- stocarea informațiilor necesare pentru urmărirea evoluției în timp a procesului
- afişare către operator
- oprirea şi repornirea în condiții de siguranță a sistemului
- alte funcții, dependente de tipul sistemului

Fiecare dintre aceste funcții au propriile constrîngeri de timp (hard, soft sau fără). De exemplu, modulele pentru controlul al diferitelor mărimi, au o constrîngere de timp în sensul că trebuie să se execute la fiecare T_r unități de timp; în practică, această valoare poate fi relaxată la o valoare de $T_r \pm e_r$ unități de timp. În mod similar se poate proceda cu perioada de achiziție, care poate fi exprimată sub forma $T_a \pm e_a$. Cerințele pot fi relaxate şi mai mult, de exemplu permițînd ca un set de date la o sută de achiziții să se piardă. Toate aceste constrîngeri vor face parte din specificare şi se va ține cont de ele în cadrul procesului de testare.

Tot în cadrul etapei de proiectare trebuie aleasă şi modalitatea în care va fi implementat efectiv software-ul; în acest sens există trei posibilităţi de abordare [Ben94]:

- ***proiectarea utilizînd un singur program*** - în cadrul acestei abordări, modulele sunt tratate ca subrutine ale unui program principal. Acest tip de structură este uşor de programat, dar impune constrîngeri severe aupra timpilor: toate modulele trebuie să se execute într-un timp mai mic decît o perioadă de timp dată. Utilizarea unui singur program se pretează la sisteme nu foarte complexe şi aceasta conduce la o proiectare simplă, utilizează minimum de software şi hardware şi este relativ uşor de testat. Dacă dimensiunea sistemului creşte, există o tendinţă în faza de proiectare detaliată de a împărţi modulele nu după funcţionalitate ci de aşa manieră încît să le permită acestora să se termine în perioada de timp specificată.

- ***proiectarea foreground/background*** - are anumite avantaje certe faţă de cea utilizînd un singur program, prin aceea că, dacă modulele cu constrîngeri de timp hard sunt separate de restul, interacţiunile dintre module sunt mai puţine, constrîngerile de timp devin mai relaxate. Modulele cu constrîngeri hard rulează în foreground, avînd o prioritate mai mare şi putînd întrerupe modulele din background. Partiţionarea foreground/background necesită suportul unui sistem de operare în timp real. Este posibil, totuşi, de a adapta şi sisteme de operare standard, ca de exemplu MS-DOS, pentru a furniza operaţii foreground/background dacă hardware-ul permite lucrul cu întreruperi. Modulele din foreground sunt astfel scrise ca rutine de întrerupere şi cele din background ca programe standard. Chiar dacă în această abordare constrîngerile de timp au fost relaxate prin această împărţire, măsurătorile care trebuiesc realizate pentru a determina performanţa sistemului sunt mult mai complicate decît în cazul unui singur program şi deci testarea devine mult mai dificilă.

- ***proiectarea multi-tasking*** - proiectarea şi implementarea sistemelor complexe de timp real este mult facilitată dacă partiţionarea foreground/background poate fi extinsă în partiţii multiple care să permită conceptul de mai multe task-uri active. Fiecare activitate este considerată, în faza de proiectare preliminară, ca fiind un task separat. Implicaţiile acestei abordări este acela că fiecare task poate să se execute în paralel, fiind necesare facilităti pentru: crearea de task-uri separate, planificare (de obicei utilizînd un sistem bazat pe priorităţi), sincronizarea task-urilor şi utilizarea în comun a datelor [Gol93]. Aceste facilităţi sunt, în mod nomal furnizate de un sistem de operare în timp real, sau de

o combinație dintre un sistem de operare în timp real şi un limbaj de programare în timp real. Din punct de vedere al verificării sistemului, după cum a fost precizat anterior, numai modelului single-task, fără întreruperi, poate să-i fie dovedită corectitudinea din punct de vedere formal; odată ce întreruperile sunt permise (ca şi în cazul multi-tasking), sistemul devine nedeterministic şi corectitudinea sa nu poate fi dovedită formal.

În finalul fazei de proiectare legăturile între hardware şi software trebuiesc stabilite: pentru cazul sistemelor în timp real este posibil să fie necesară revizuirea deciziilor anterioare referitoare la tipul de structură de calculatoare folosită, sau, în cazul unui sistem în timp real distribuit, trebuie decis asupra numărului şi tipului de procesoare, a sistemelor de comunicație, etc. Strategia de control ar putea de asemenea necesita să fie revizuită funcție de algoritmii de control care urmează să fie utilizați (întoarcere în modelul cascadă).

1.3.4.4. Implementarea software/hardware

Etapa de implementare se ocupă cu realizarea efectivă atît a structurii hardware cît şi software, precum şi cu testarea acestora.

Din punct de vedere hardware, proiectarea detaliată se ocupă cu rezolvarea problemelor privind următoarele puncte:

- alegerea efectivă a tipului de calculator (sau calculatoare) necesare
- dacă se vor utiliza plăci separate pentru intrările analogice respectiv digitale sau toate intrările vor fi concentrate pe o singura placă (aici se va lua în considerare, funcție de tipul de calculator ales, ce echipamente pentru I/O sunt compatibile şi pot fi folosite); acelaşi lucru referitor la ieşirile analogice şi digitale; funcție de numărul acestora, care este numărul de plăci de I/O necesare?
- pentru sistemele distribuite, ce tip de structură de magistrală trebuie utilizată?
- funcție de de senzorii şi actuatorii utilizați, este sau nu necesară utilizarea unor echipamente de interfațare cu plăcile de I/O?

Din punct de vedere software, etapa de implementare se ocupă de descompunerea în module respectiv proiectarea şi implementarea internă a modulelor. Descompunerea se bazează în principal pe identificarea activităților înrudite şi pe utilizarea metodologiilor general acceptate după care această descompunere poate fi realizată:

- ***descompunere funcțională*** - fiecare modul realizează o funcție specifică

- ***ascunderea informației*** - cît mai multă informație utilizată de modul să fie ascunsă în cadrul acestuia [GJM91]; practic interfața cu exteriorul să fie redusă la minimum
- ***object-oriented*** - împărțirea sistemului în entități care conțin atît date cît şi funcții care operează asupra datelor; este o completare a conceptului de ascundere a informației prezentat mai sus.

În plus, pentru sistemele timp real, sunt necesare criterii suplimentare care trebuiesc luate în considerare la împărțirea sistemului în module, funcție de tipurile de constrîngeri de timp existente: module cu constrîngeri de timp hard, module cu constrîngeri de timp soft, module interactive [KS97] [Kop97]. Ideea este să se minimizeze pe cît posibil cantitatea de software care intră în categoria constrîngerilor de timp hard, întrucît acest tip de module sunt cele mai dificil de implementat şi testat.

Importanța separării activităților în sistemele în timp real, în task-uri de timp real şi task-uri care nu sunt de timp real, precum şi împărțirea ulterioară a task-urilor de timp real în cele două tipuri (hard şi soft), rezultă din diferitele niveluri de dificultate care apar în proiectarea şi implementarea acestor tipuri diferite de parți de program. Studiile experimentale arată că, în mod deosebit programele care implică timp real precum şi diverse operații de interfațare sunt mult mai dificil de construit decît programele standard de procesare a datelor.

1.3.4.5. Testarea

Un ultim aspect în cadrul procesului de implementare îl constituie testarea. ***Testarea dinamică*** este metoda convențională pentru verificarea aplicațiilor în timp real. Spre deosebire de sistemele informaționale, în cazul sistemelor de timp real este necesară realizarea testării dinamice a aplicației realizate, în cadrul tuturor etapelor de dezvoltare [Lau93], conform modelului prezentat în Figura 8.

Simularea reprezintă o unealtă fundamentală în cadrul procesului de dezvoltare a sistemelor de control timp real: validarea strategiei de control adoptate este realizată întotdeauna utilizînd simularea, existînd o mulțime de medii utilizate în acest scop, cel mai cunoscut şi general utilizat la ora actuală fiind Matlab/Simulink. De asemenea, simularea poate fi utilă şi pentru realizarea unei mai profunde cunoaşteri a dinamicii sistemului şi în consecință la o evaluare generală a caracteristicilor de tip real a sistemului în cauză.

După cum se poate observa din Figura 8, numai în ultimul pas (4) testarea se face în condiții reale, aplicația realizată fiind legată de sistemul fizic real, în rest se utilizează un model simulat al sistemului. Strategia de testare trebuie foarte bine definită, astfel încît ea să includă testarea atît la condiții de funcționare normală cît şi anormală a sistemului controlat pentru toate fazele de

dezvoltare, de la specificare pînă la implementarea propriu-zisă. Un aspect foarte important îl reprezintă, în procesul de testare, alegerea corespunzătoare a scenariilor de test, astfel încît acestea să acopere toate situațiile posibile.

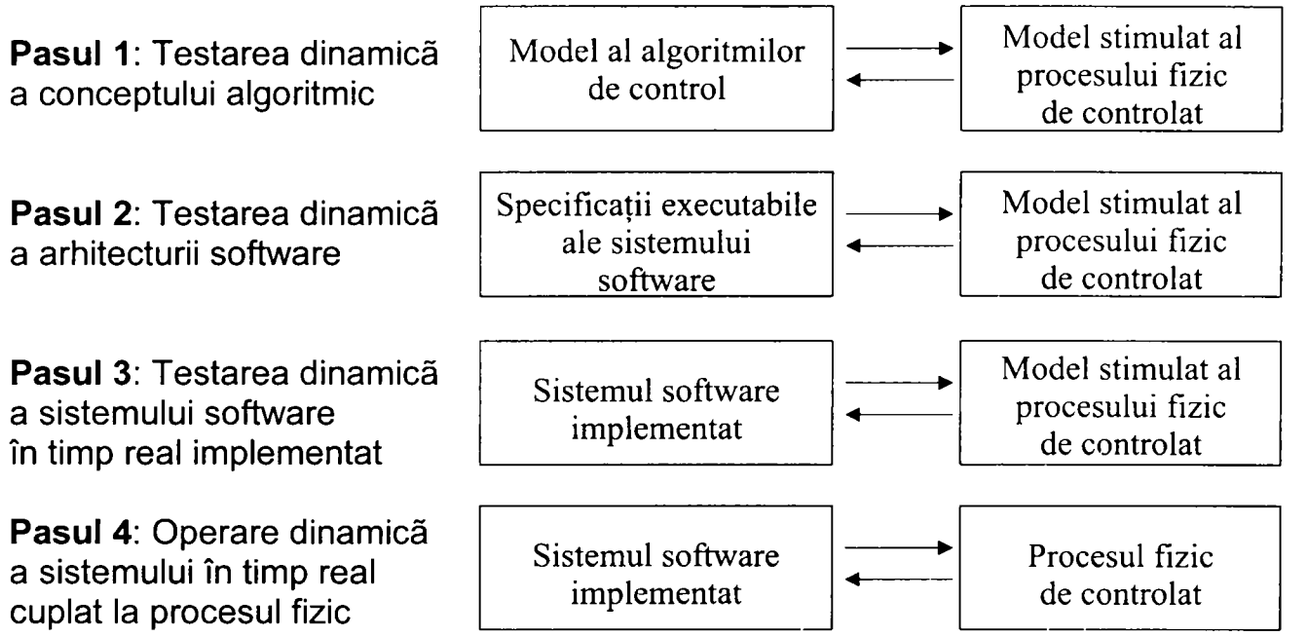


Figura 8 : Secvența de testare dinamică din patru paşi

Pentru sistemele timp real complexe acest obiectiv este însă aproape imposibil de realizat, astfel încît se preferă utilizarea metodelor formale de verificare [LH95]. Acestea reprezintă cea mai riguroasă tehnică de analiză şi verificare a programelor, însă este foarte laborioasă în cazul aplicării pentru sistemele complexe; astfel de analize sunt însă justificate pentru sistemele în timp real critice, în cadrul cărora testarea dinamică nu poate demonstra că aplicația funcționează corect în toate circumstanțele. Totuşi, analiza formală nu reprezintă un substitut pentru testarea dinamică: de exemplu erorile hardware sau de compilare pot fi uşor depistate prin intermediul acesteia.

1.4. Structura tezei

Teza este structurată pe şapte capitole care formează trei părți distincte.

In prima parte (capitolele 1-5) s-a urmărit prezentarea detaliată a ***contribuțiilor teoretice*** care stau la baza lucrării. In cea de-a doua parte (capitolul 6) s-a ilustrat, prin intermediul unui studiu de caz, o modalitate de aplicare în practică a noțiunilor teoretice prezentate în capitolele anterioare, această parte constituind ***contribuția practică*** a tezei. In final, în cea de-a treia parte (capitolul 7) au fost sintetizate principalele ***concluzii***, accentuate contribuțiile originale şi trasate principalele direcții ulterioare de dezvoltare.

Astfel, primul capitol începe cu prezentarea caracteristicilor de bază ale sistemelor timp real, împreună cu o sumară clasificare a acestora. Modelele existente din cadrul ingineriei controlului automat **ICA** şi a ingineriei programării **IP** referitoare la ciclul de viață al software-ului sunt analizate din perspectiva aplicabilității acestora pentru sistemele timp real. Avînd în vedere caracterul interdisciplinar al acestor sisteme, s-a demonstrat că este necesară utilizarea combinată atît a modelelor din domeniul **ICA**, cît şi a celor din domeniul **IP**. Pe această bază, a fost conceput şi prezentat un model general al ciclului de viață al sistemelor timp real.

Capitolul al doilea prezintă principalele tipuri de modele care pot fi utilizate în dezvoltarea aplicațiilor timp real; chiar dacă fiecare dintre acestea reprezintă diferite abstracții ale realității, avînd grade diferite de formalism, este subliniată ideea că aceleaşi concepte sunt necesare în construirea modelului, indiferent de tipul acestuia. Dintre acestea, un rol deosebit în cadrul aplicațiilor timp real îl au proprietățile temporale, care trebuiesc în mod obligatoriu captate în model.

In plus, capitolul al doilea rafinează din ce în ce mai mult clasificarea sistemelor timp real din cadrul primului capitol, prezentînd o comparație între sistemele ***time-triggered*** **TT** şi cele ***event-triggered*** **ET**. Caracteristicile fiecărui tip de sistem sunt prezentate, împreună cu avantajele şi dezavantajele acestora. Este subliniat de asemenea faptul că, cerințele de predictabilitate şi determinism, specifice sistemelor timp real critice sunt practic realizabile în cazul utilizării variantei ***time-triggered***; varianta ***event-triggered*** avînd în schimb avantajul flexibilității.

In plus față de limbajele de programare şi metodologiile de proiectare, dezvoltarea eficientă a sistemelor timp real implică şi utilizarea unor unelte şi medii CASE care să asiste procesul de dezvoltare de-a lungul tuturor etapelor acestuia. Capitolul al treilea prezintă pe scurt o selecție a celor mai răspîndite unelte şi medii CASE existente la ora actuală şi care pot fi utilizate pentru proiectarea şi dezvoltarea aplicațiilor timp real, împreună cu avantajele şi

dezavantajele acestora. Facilitățile puse la dispoziție de fiecare unealtă în parte sunt prezentate şi analizate.

Posibilitatea aplicării tehnologiilor obiectuale în dezvoltarea aplicațiilor timp real este investigată; din prisma utilizării acestora în aplicațiile timp real se ilustrează de asemenea principalele probleme pe care tehnologiile obiectuale şi uneltele CASE aferente momentan nu le rezolvă. Ca şi o concluzie, se prezintă în final o listă de facilități pe care ar trebui să le includă o unealtă CASE ideală astfel încît ea să poată fi utilizată cu succes în proiectarea şi dezvoltarea oricărei aplicații timp real.

Capitolul al patrulea prezintă caracteristicile hardware şi software a controller-elor programabile (PLC-uri) în contextul utilizării acestora în aplicațiile de timp real. In urma analizei modului de operare ale acestora, este ilustrată posibilitatea utilizării paradigmei ciclice (definită sub numele de **paradigma Cy-Clone**), şi care se bazează pe principiul ***resource adequacy***, pentru dezvoltarea sistemelor de control timp real utilizînd PLC-uri. S-a demonstrat că o astfel de combinație este relativ ***simplu*** de realizat practic şi ea are, datorită existenței unei discipline temporale stricte (sistem **time-triggered**), avantajul ***predictabilității***, element esențial pentru sistemele critice.

De asemenea, în urma utilizării **paradigmei Cy-Clone**, s-a demonstrat că se poate obține o mai bună detectare şi localizare a erorilor, şi implicit o creştere a ***dependabilității*** sistemului. Utilizarea combinată a PLC-urilor şi a paradigmei ciclice respectă deci criteriile de de bază care ar trebui să stea la baza dezvoltării oricărui sistem de control timp real, şi care pot fi sintetizate cu ajutorul sintagmei ***simplitate => predictabilitate => dependabilitate***, prezentată în cadrul capitolului 1.1.

Metodologia propusă pentru dezvoltarea aplicațiilor de control timp real utilizînd PLC-uri şi abordarea ciclică este descrisă pe larg în capitolul al cincilea. Descrierea acesteia începe cu o prezentare generală a metodologiei împreună cu a tuturor conceptelor utilizate; se definesc şi se rafinează în acest sens noțiunile de ***model prescriptiv*** şi ***model descriptiv***. Ulterior, fiecare etapă a metodologiei este tratată în mod distinct, definindu-se în mod clar ***intrările***, ***ieşirile*** precum şi ***paşii*** care trebuie urmați în cadrul fiecărei etape.

Astfel, este ilustrat faptul că, utilizînd metodologia propusă, ***garantarea*** performanțelor în toate situațiile poate fi realizată, datorită predictabilității obținute în urma utilizării paradigmei **Cy-Clone**. De asemenea, se mai arată că metodologia propusă poate fi foarte uşor adaptată pentru o largă categorie de aplicații complexe din domeniul controlului automat timp real, şi că ea furnizează nu numai un set de concepte şi criterii care pot fi utilizate în procesul de proiectare şi dezvoltare, dar şi o combinație de unelte software prin intermediul cărora să se realizeze aplicarea în practică într-o manieră simplă şi eficientă.

In finalul capitolului sunt prezentate principalele avantaje dar şi dezavantaje ale metodologiei propuse. Se poate observa astfel că avantajele primează în mod clar; astfel concluzia care se impune este aceea că metodologia propusă reprezintă o soluție viabilă pentru o largă categorie de aplicații timp real, şi în mod deosebit pentru aplicațiile de conducere şi control a proceselor industriale.

In scopul de a valida din punct de vedere practic fundamentele teoretice care au stat la baza metodologiei prezentate în capitolul 5, şi care au fost la rîndul lor prezentate pe larg în capitolele anterioare (1-4), în partea a doua a lucrării s-a prezentat un exemplu de aplicare a acestora. Astfel, în cadrul capitolului al şaselea, este ilustrat un studiu de caz pentru o aplicație timp real care a fost dezvoltată utilizînd metodologia propusă.

Aplicația ***Sistemul automat de monitorizare şi control a instalației geotermale de la Universitatea din Oradea*** constituie un exemplu concludent a modului de utilizare a metodologiei propuse în cazul unei aplicații concrete şi complexe în acelaşi timp. Modul de utilizare al metodologiei precum şi uneltele CASE aferente fiecărei etape sunt descrise în detaliu. Prin acest exemplu, metodologia prezentată în cadrul capitolului 5 îşi dovedeşte pe deplin oportunitatea şi utilitatea.

In final, în cadrul capitolului al şaptelea, au fost sintetizate principalele concluzii evidențiindu-se în principal contribuțiile originale existente de-a lungul tezei; au fost identificate de asemenea şi principalele direcții de dezvoltare de viitor, subliniindu-se faptul că prezenta lucrare reprezintă doar o etapă din cadrul unui proiect de lungă durată.

2. SISTEME TIMP REAL *TIMED-TRIGGERED* (TT) VERSUS *EVENT TRIGGERED* (ET)

2.1. Clasificarea sistemelor timp real

Sistemele timp real pot fi clasificate din diferite puncte de vedere [KS97][BF97]. Astfel, aceste clasificări pot fi realizate în funcție de caracteristicile aplicației, şi în consecință sunt dependente de factori externi sistemului de calcul utilizat sau în funcție de caracteristicile implementării şi proiectării propriu-zise, caz în care sunt dependente de factori interni sistemului de calcul.

2.1.1. Sisteme hard real-time versus soft real-time

O binecunoscută clasificare împarte sistemele de timp real în:

- **sisteme hard real-time**: în cadrul acestora trebuie să se garanteze prin proiectare satisfacerea constrîngerilor de timp în orice situație posibilă; utilitatea unor astfel de sisteme depinde tocmai de posibilitatea de a asigura o performanță predictibilă în decursul unor situații limită. Nesatisfacerea constrîngerilor de timp în cazul sistemelor hard real-time conduce la compromiterea funcționării corecte a sistemului
- **sisteme soft real-time**: acestea sunt sisteme pentru care o eventuală nesatisfacere a constrîngerilor de timp nu compromite corectitudinea acestuia

Un exemplu tipic de control **hard real-time** este bucla de control a temperaturii unui amestec într-o instalație chimică, cu ajutorul unei valve care închide/deschide circuitul de aerisire. În termeni de control, pentru proiectarea unui algoritm potrivit buclei de control a temperaturii, se impune alegerea unui interval de achiziție T_s corespunzător, astfel încît, în cadrul acestuia să fie posibilă realizarea operațiilor de citire a intrării, de realizare a calculelor de control precum şi de a transmite în afară, către actuatorul care acționează valva, ieşirea calculată.

Practic, proiectarea unui sistem timp real hard, care trebuie să producă rezultate corecte la momentul corect, este fundamental diferită de proiectarea unui sistem în timp real soft. Diferențele pot fi sumarizate conform Tabelului 1:

Tabelul 1: Sisteme timp real hard versus sisteme timp real soft

Caracteristica	*Sisteme hard*	*Sisteme soft*
Timp de răspuns	obligatoriu	dorit
Performanța la încărcare maximă	predictibilă	degradabilă
Sursa controlului	mediul	calculatorul
Siguranța în funcționare	critică, de cele mai multe ori	ne-critică
Dimensiunea fişierelor de date	mică sau medie	mare
Tipul redundanței	activă	checkpoint-recovery
Integritatea datelor	pe termen scurt	pe termen lung
Detecția erorilor	autonomă	asistată de utilizator

Astfel, cerințele pentru timpul de răspuns în cadrul sistemelor ***hard real-time*** sunt de multe ori de ordinul a cîtorva milisecunde sau chiar mai puțin, excluzînd practic posibilitatea intervenției utilizatorului uman în situații critice. In acest caz sistemul trebuie să fie autonom pentru a asigura operarea în siguranță a procesului. Pentru sistemele ***soft real time*** şi ***on-line*** cerințele de timp sunt mai relaxate; în plus, nesatisfacerea unei constrîngeri de timp nu compromite funcționarea corectă şi sigură a sistemului [Kop97].

Totuşi, în practică, clasificarea de mai sus este uneori orientativă: de exemplu, în anumite circumstanțe, o ratare ocazională a unei constante hard periodică (ca cea din exemplul de mai sus) nu este o problemă serioasă: într-o buclă de control **(feed-back)**, pierderea ocazională a datelor unei achiziții nu va produce disturbanțe majore în sistemul de controlat, atîta timp cît variabila manipulată rămîne la valoarea anterioară. Din acest motiv, în cadrul specificațiilor unui astfel de sistem, constantele de timp aferente pot fi relaxate într-o oarecare măsură (Tabelul 2).

Astfel, constrîngerile de timp pot fi definite formal conform Tabelului 2 [Ben94], în cadrul căruia:

$t_c(i)$ = intervalul între ciclul **i** şi **i-1**

$t_e(i)$ = timpul de răspuns la a-**i**-a apariție a evenimentului **e**

t_s = intervalul periodic (ciclic) dorit

T_e = timpul maxim de răspuns permis pentru evenimentul **e**

T_a = timpul mediu de răspuns permis pentru evenimentul **e**, măsurat pe un interval T

n = numărul de apariții al evenimentului e în intervalul de timp T, respectiv numărul de repetiții ciclice de-a lungul intervalului de timp T

a = o toleranță mică de timp

Tabelul 2: Definirea formală a constrîngerilor de timp

Hard		***Soft***	
Periodic (ciclic)	**Aperiodic (event)**	**Periodic (ciclic)**	**Aperiodic (event)**
$t_C(i) = t_S \pm a$	$t_e(i) \le T_e$	$\frac{1}{n}\sum_{i=1}^{n} t_c(i) = t_s \pm a$ $n = \frac{T}{t_s}$	$\frac{1}{n}\sum_{i=1}^{n} t_e(i) \le T_a$ $n = \frac{T}{t_s}$

Totuşi, constantele de timp hard, în mod evident reprezintă constrîngeri de de timp mult mai severe, cu implicații mult mai mari asupra sistemelor în timp real, decît cele soft. Majoritatea sistemelor timp real conțin un amestec de activități avînd constrîngeri de timp atît hard cît şi soft; de asemenea pot exista activități care nu sunt de timp real. Proiectarea unui astfel de sistem implică recunoaşterea fiecărui tip de activitate, şi, pe cît posibil, gruparea şi reducerea numărului de task-uri care prezintă constrîngeri de timp real.

2.1.2. Sisteme fail-safe versus fail-operational

Pentru anumite categorii de sisteme în timp real pot fi identificate anumite stări aşa numite **de siguranță** care pot fi atinse în cazul apariției unei defecțiuni. Dacă sistemul poate ajunge într-o astfel de stare imediat după apariția unei defecțiuni, atunci acesta este ***fail-safe***. Aceasta este o caracteristică externă sistemului de calcul propriu-zis şi nu în toate aplicațiile este posibilă atingerea unei astfel de stări de siguranță.

In multe sisteme în timp real există un dispozitiv special, aşa numitul **watchdog**, care este utilizat pentru monitorizarea operării sistemului de calcul. Sistemul de calcul trebuie să transmită periodic un semnal (o ieşire digitală de o formă predefinită) către watchdog, în intervalul de timp stabilit.

Dacă acest semnal nu ajunge la watchdog la timp, acesta presupune că a avut loc o defecțiune la sistemul de calcul şi forțează obiectele controlate într-o stare de siguranță. In astfel de sisteme satisfacerea cerințelor de timp este necesară doar pentru a controla disponibilitatea acestuia, nu şi pentru siguranță, deoarece sistemul ajunge într-o stare sigură în cazul apariției unei defecțiuni.

In cazul în care sistemul este de aşa natură încît starea de siguranță nu poate fi atinsă, în cazul unei defecțiuni sistemul trebuie să furnizeze un nivel minim de operare, astfel încît consecințele defecțiunii să nu fie catastrofale. Astfel de sisteme sunt ***fail-operational***.

2.1.3. *Răspuns garantat* versus *cel mai bun efort*

Dacă încă din faza de proiectare a sistemului se poate face o verificare analitică a răspunsului sistemului care să dovedească corectitudinea acestuia în orice situație posibilă, avem de-a face cu un sistem cu ***răspuns garantat***. Probabilitatea de defecțiune a unui sistem perfect cu răspuns garantat este limitată la probabilitatea cu care presupunerile referitoare la situațiile de încărcare maximă şi la numărul şi tipul defecțiunilor posibile sunt în concordanță cu realitatea. Un sistem cu răspuns garantat necesită o planificare şi o proiectare extrem de atent realizată, combinată cu o analiză extensivă în timpul fazei de proiectare.

Dacă o astfel de garanție obținută în mod analitic nu poate fi dată, atunci avem o proiectare caracterizată prin ***cel mai bun efort***. In cazul acesteia nu este necesară specificarea riguroasă a ipotezelor de încărcare maximă şi de defecțiune, suficiența proiectării va fi stabilită în urma fazelor de testare şi integrare a sistemului. Este foarte dificil de stabilit că un sistem construit pe baza principiului ***cel mai bun efort*** va opera corect în toate situațiile posibile, mai ales în cele care apar cu o probabilitate foarte redusă. In prezent însă multe sisteme în timp real ne-critice sunt proiectate conform acestei paradigme.

2.1.4. *Resource-adequate* versus *resource-inadequate*

Sistemele cu răspuns garantat se bazează pe principiul ***resource-adequacy*** care stipulează faptul că există suficiente resurse disponibile pentru a face față inclusiv situațiilor de încărcare maximă sau de defecțiune apărute. Multe sisteme timp real ne-critice se bazează pe principiul ***resource-inadequate*** în sensul că asigurarea resurselor suficiente pentru tratarea tuturor situațiilor posibile nu este o soluție viabilă din punct de vedere economic şi că o strategie de alocare dinamică bazată pe folosirea în comun a resurselor (alocate într-o manieră probabilistică) este acceptabilă.

Pentru sistemele timp real viitoare se tinde însă către utilizarea paradigmei ***resource-adequacy***, mai ales în cazul sistemelor timp real hard. Utilizarea calculatoarelor în aplicații cu mari volume de date şi foarte complexe forțează proiectantul să garanteze funcționarea corectă a sistemului în toate situațiile posibile. Deci, sistemele în timp real hard trebuie proiectate conform paradigmei ***raspuns garantat*** care poate fi asigurată numai pe baza principiului ***resource-adequacy***.

2.1.5. Sisteme *event-trigered* (ET) versus *timed-triggered* (TT)

Evoluția sistemelor timp real poate fi modelată utilizînd o linie abstractă de timp care semnifică trecerea timpului din trecut înspre viitor. Orice apariție care taie această linie imaginară este un eveniment; de exemplu un ceas digital partiționează linia timpului într-o secvențe de timp de durate egale (perioada ceasului).

Un **trigger** este un eveniment care are ca rezultat execuția unui task sau transmiterea unui mesaj. In funcție de mecanismul adoptat pentru inițierea comunicării şi activitatății de procesare, există două abordări distincte care pot fi utilizate în procesul de proiectare a aplicațiilor timp real[:

- abordarea ***event-triggered (ET)*** – în cadrul acesteia toate activitătile din cadrul sistemului precum şi comunicarea în sistem se realizează în urma apariției unor evenimente. Sesizarea evenimentelor semnificative se realizează utilizînd binecunoscutul mecanism al întreruperilor şi în consecință este necesară aplicarea unei strategii de planificare dinamică pentru activarea task-ului corespunzător tratării evenimentului apărut

- abordarea ***timed-triggered (TT)*** – în cadrul căreia toate activitățile, inclusiv comunicarea, sunt inițiate la momente predefinite de timp, existînd o singură întrerupere, şi anume întreruperea de ceas. Dacă sistemul este distribuit atunci se presupune că ceasurile fiecărui nod sunt sincronizate între ele conform unui ceas global şi că fiecare eveniment poate fi ***ştampilat*** **(timestamped)** în funcție de acest timp. Granularitatea ceasului global trebuie astfel aleasă încît ordinea în timp a apariției evenimentelor într-un sistem distribuit să poată fi stabilită pe baza ***şampilei de timp*** a acestora.

De exemplu, în cazul sistemelor timp real încorporate, calculatorul este doar un element funcțional din cadrul sistemului în ansamblul său. O caracteristică comună a acestei categorii de sisteme este aceea că, calculatoarele sunt conectate la procese care sunt exterioare calculatorului printr-o mare varietate de dispozitive de interfațare care recepționează şi transmit o mare varietate de stimuli.

Aceste procese operează în propria lor scară de timp, iar despre calculator se spune că operează în timp real dacă acesta se aliniază cu viteza de desfăşurare a proceselor.

Astfel, dacă sincronizarea dintre procesele externe şi task-urile interne este definită în funcție de timpul curent, înseamnă că la proiectarea sistemului în cauză a fost utilizat mecanismul **TT**; dacă însă aceasta se defineşte în funcție de anumite evenimente, s-a utilizat abordarea **ET**. In practică există şi o a treia categorie de sisteme, aşa numite ***interactive***, în cadrul cărora relațiile între acțiunile din cadrul computerului şi cele externe sunt mai puțin strict definite [Kop91].

Trebuie în mod obligatoriu precizat faptul că, pentru sistemele care abordează modalitatea **event-triggered ET** şi lucrează prin întreruperi, predictibilitatea este un deziderat greu de atins. De asemenea, o altă cerință, şi mai greu de atins dar esențială pentru sistemele critice, aceea de determinism, nu poate fi practic realizată. De altfel, orice modalitate de a schimba în mod dinamic (de exemplu, conform priorității) ordinea proceselor în astfel de sisteme rezultă într-un sistem nedeterministic. Rezultatele obținute de Liu şi Layland (1973) [SR90] privind algoritmul de planificare rate-monotonic pot fi utilizate doar pentru dovedirea predictibilității; ele însă nu rezolvă problema determinismului.

2.2. Aspecte teoretice privind modelarea sistemelor timp real

Avantajele şi dezavantajele abordărilor **TT** versus **ET** pentru sisteme timp real hard trebuiesc bine cunoscute în cazul modelării unui astfel de sistem; funcție caracteristicile intrinseci ale aplicației propriu-zise, se alege abordarea care se pretează cel mai bine în contextul performanță/cost. Indiferent însă de calea aleasă, un model al sistemelor în timp real trebuie să se bazeze pe un set general valabil şi bine definit de concepte.

2.2.1. Elemente esențiale pentru construirea modelului unui sistem timp real

Pentru acelaşi sistem pot fi construite diverse modele:

- modelul fizic
- modelul simulat a procesului
- modelul matematic a diverselor fenomene fizice
- modelul exprimat în logică formală a sistemului

Toate aceste modele reprezintă diferite abstracții ale realității şi au un diferit grad de formalism. Modelele formale au avantajul unei notații precise şi a unor reguli foarte riguroase care suportă verificarea automată a anumitelor proprietăți ale sistemului modelat.

Esențialul în construirea unui model stă în corectitudinea şi simplitatea acestuia. Aceste cerințe sunt în general contradictorii: reducerea numărului de informații sau abstractizarea în vederea simplificării modelului poate fi făcută numai în cazul în care corectitudinea acestuia nu este afectată [How93]. De asemenea, toate presupunerile făcute în procesul de modelare în scopul simplificării trebuie specificate clar întrucît acestea definesc practic validitatea acestuia. Probabilitatea ca presupunerile făcute să fie reale (***assumption coverage***) limitează probabilitatea ca şi concluziile derivate din cadrul modelului să fie valide în realitate. Un exemplu în acest sens este ilustrat de proprietățile enunțate pentru modelele $\mathbf{M_d}$ şi $\mathbf{M_p}$ definite în cadrul paragrafului 5.2.

Pentru sistemele timp real critice următoarele elemente esențiale trebuiesc definite în cadrul modelului [Irw96]:

- ***situațiile limită*** - adică situația de încărcare maximă precum şi toate cazurile de defecțiune care pot apărea
- ***proprietățile temporale*** care trebuie în mod obligatoriu să fie incluse în model

Din punct de vedere al proprietăților temporale acestea se referă în primul rînd la durata (sau timpul de execuție) asociată unei anumite acțiuni. Astfel, pentru o acțiune a dată există următoarele mărimi cantitative care să descrie comportarea în timp a acesteia:

- ***durata actuală*** **sau** ***timpul de execuție*** $d_{act}(a,x)$, reprezentînd numărul de unități de timp necesare din momentul în care acțiunea *a* este declanşată pînă în momentul cînd aceasta se termină pentru setul de date de intrare *x*
- ***durata minimă*** $d_{min}(a)$, reprezentînd cel mai mic interval de timp necesar terminării acțiunii *a,* măsurat asupra tuturor datelor de intrare posibile
- ***timpul de execuție*** în cazul cel mai defavorabil $d_{WCET}(a)$, reprezentînd cel mai lung timp necesar terminării acțiunii a în condiții de încărcare maximă şi defecțiuni, măsurat asupra tuturor posibilităților datelor de intrare
- ***timpul de jitter***, reprezentînd diferența între timpul de execuție în cazul cel mai defavorabil şi durata minimă, $d_{WCET}(a) - d_{min}(a)$

O altă caracteristică temporală, şi anume ***frecvența activărilor*** unei anumite acțiuni per unitatea de timp, trebuie avută în vedere la definirea unui model pentru un sistem timp real. Aceasta deoarece fiecare resursă din cadrul sistemului are o capacitate finită determinată de parametri acesteia; aceasta poate să-şi îndeplinească cu succes obligațiile din punct de vedere temporal numai dacă frecvența activărilor resursei respective se află sub un control strict.

În vederea evitării pe cît posibil a încărcării excesive a modelului există însă şi o serie de aspecte care nu este necesar să fie incluse în modelul creat. Dintre acestea, cele mai semnificative se referă la detalii de reprezentare a datelor, modul efectiv de implementare a algoritmilor de control şi de transformare a datelor, etc.

2.2.2. Control temporal versus control logic

Astfel, într-un sistem ***event-triggered*** **ET** se utilizează mecanismul întreruperilor pentru tratarea evenimentelor externe. Cînd un astfel de mecanism este activat şi apare o întrerupere, execuția task-ului curent este întreruptă şi hardware-ul forțează o schimbare de context către rutina de tratare a întreruperii corespunzătoare. După terminarea tratării întreruperii, o nouă schimbare de context trebuie realizată fie pentru a continua task-ul întrerupt, fie pentru a trata o altă întrerupere apărută între timp. Aceste schimbări de context consumă un anumit timp, aşa numitul **WCAO (Worst Case Administrative Overhead)**. Apariția fiecărei întreruperi reduce capacitatea unității centrale disponibilă aplicației, după cum este ilustrat în Figura 9. Acest lucru este valabil şi în cazul în care rutina de tratare decide că întreruperea a fost eronată şi nu activează nici un alt task.

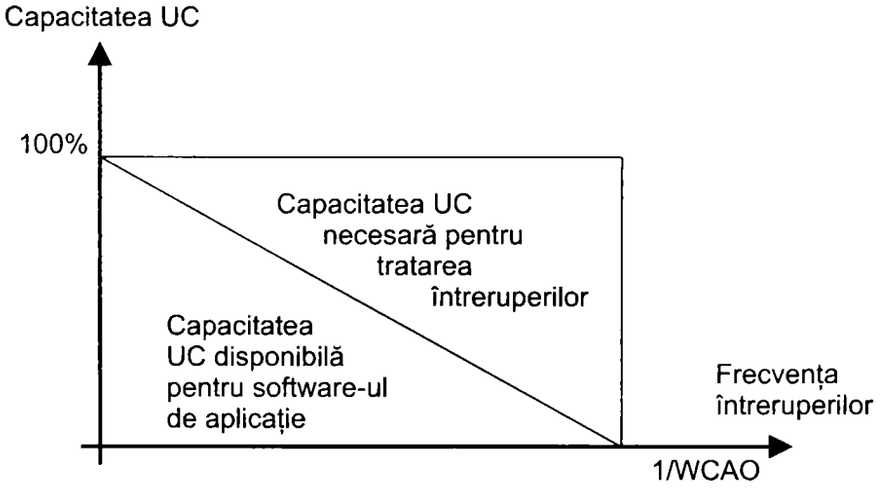


Figura 9: Capacitatea CPU versus 1/WCAO

Se observă din Figura 9 că dacă frecvența întreruperilor atinge valoarea 1/WCAO, capacitatea unității centrale disponibilă pentru aplicații devine zero; este deci de o importanță covîrşitoare ca frecvența întreruperilor într-un sistem timp real să fie limitată. Acest lucru este practic imposibil într-un sistem ET, deoarece sursa întreruperilor se află în afara sferei de control a calculatorului.

In cazul unui sistem timp real ***timed-triggered*** **TT** însă, controlul rămine întotdeauna în cadrul calculatorului [Kop92][Kop97]. Pentru a sesiza un anumit eveniment exterior, un sistem **TT** trebuie să utilizeze un task periodic **time-triggered** care să evalueze starea variabilelor din sistem. Perioada acestui task trebuie să fie mai mică decît **laxity**-ul (diferența între deadline şi timpul de execuție) a oricărei tranzacții în timp real care poate fi activată de un anumit eveniment. Şi utilizarea unui task trigger generează timpi suplimentari în sistem; practic, dacă **laxity**-ul unei tranzacții este foarte mic (<1ms), overhead-ul asociat task-ului trigger este mult prea mare pentru a fi admisibil.

Există la ora actuală numeroase sisteme timp real care utilizează atît abordarea **ET** cît şi abordarea **TT**. Totuşi, majoritatea arhitecturilor tind să favorizeze mai mult una dintre ele: semnalele de control sunt predominant event-triggered sau predominant time-triggered. O comparație între cele două variante poate fi sumarizată conform Tabelului 3.

Tabelul 3: Activarea task-urilor prin întreruperi versus utilizarea unui task trigger

Caracteristica	**Intrerupere**	**Task trigger**
Factori care afectează latența	răspunsul la întrerupere	perioada de achiziție
Sursa controlului	externă	internă
Overhead-ul datorat scanării	nu există	WCET(laxity_trigger) / laxity
Overhead-ul datorat procesării	variabil	constant
Posibilitatea preemțiunii	oricînd	controlată
Condițiile de trigger	simple	complexe
Predictabilitate	nivel scăzut	nivel ridicat

2.2.3. WCET (Worst-Case Execution Time)

Văzută din exterior, o aplicație timp real distribuită poate fi partiționată într-un set de cluster-e care comunică între ele. Fiecare cluster poate fi împărțit la rîndul său în unități tolerante la defecțiuni **FTU (Fault Tolerant Unit)** legate în rețele locale şi care conțin ca şi noduri unul sau mai multe calculatoare. Pe fiecare astfel de calculator se pot executa în mod concurent mai multe task-uri.

Elementul structural de bază în cadrul unui astfel de sistem distribuit îl reprezintă ***task***-ul. Durata actuală a task-ului care rulează pe o anumită maşină cu setul de date de intrare x, este dată de parametrul $d_{act}(task,x)$. Dacă nu există puncte de sincronizare cu alte task-uri în cadrul unui task, acesta îşi poate continua execuţia pînă la sfîrşit; în acest caz timpul de execuţie poate fi determinat în mod izolat deoarece acesta nu depinde în mod direct de modul de execuţie al altor task-uri. În caz contrar, timpul de execuţie pentru cazul cel mai defavorabil reprezintă o problemă globală a sistemului şi estimarea acestuia nu este întotdeauna posibilă.

Garantarea deadline-ului pentru o tranzacţie în timp real poate fi realizată numai în cazul în care timpii de execuţie pentru cazul cel mai defavorabil **WCET** pentru toate task-urile care sunt parte a tranzacţiei sunt cunoscuţi dinainte [Kop97]. Aceşti timpi trebuie de asemenea să fie valabili pentru toate datele de intrare şi pentru toate scenariile de execuţie posibile. În plus, mai trebuie luate în considerare şi întîrzierile de ordin administrativ, **WCAO**, cauzate de schimbări de context, planificare, reîncărcarea cache-ului sau DMA.

Pentru un task simplu, care rulează pe un hardware dedicat, fără preemţiune şi fără a necesita serviciile sistemului de operare, **WCET** depinde în principal de următorii factori:

- codul sursă al task-ului
- proprietăţile codului obiect generat de compilator
- caracteristicile hardware pe care rulează

În prezent, stabilirea prin metode analitice de calcul a timpului **WCET** pentru un astfel de task simplu, poate fi realizată numai în anumite condiţii restrictive. De exemplu, referitor la codul sursă al task-ului, în cazul utilizării unui limbaj de nivel înalt trebuie să fie posibil să se evalueze timpul maxim de execuţie pentru toate construcţiile de limbaj; acest lucru s-ar traduce în absenţa apelurilor recursive, a ciclurilor nelimitate (unbounded) şi a structurilor de date dinamice.

De asemenea, compilatorul trebuie să includă anumite informaţii specifice aplicaţiei în cadrul codului obiect pentru ca analiza **WCET** a codului să fie posibilă. În ceea ce priveşte însă task-urile complexe, care se sincronizează cu alte task-uri şi care pot fi întrerupte, o analiză **WCET** a acestora este în acest moment practic imposibil de realizat. Totuşi, întrucît stabilirea anumitelor limite pentru **WCET** este absolut necesară în aplicaţiile practice, această problemă importantă este în prezent realizată prin combinarea a diverse tehnici în diferitele faze de dezvoltare a aplicaţiei. Cele mai importante se referă la testarea unei implementări pentru a obţine date experimentale asupra timpilor **WCET**; în final, testarea exhaustivă a implementării finale va valida (eventual) presupunerile anterioare şi va stabili marginea de siguranţă între timpii **WCET** presupuşi şi cei măsuraţi.

2.3. Concluzii

In cadrul acestui capitol s-a demonstrat că principalul element care trebuie considerat în procesul de proiectare al unui sistem de control timp real îl reprezintă posibilitatea garantării satisfacerii tuturor constrîngerilor de timp impuse. Pentru a putea realiza acest lucru, modelarea unor astfel de sisteme trebuie să aibă la bază paradigma ***răspuns garantat***, ceea ce implică asigurarea tuturor resurselor necesare, ***resource adequacy***. In plus, corectitudinea şi simplitatea modelului sunt elemente esențiale în construirea acestuia, cu condiția ca reducerea numărului de informații sau abstractizarea în vederea simplificării să nu afecteze corectitudinea acestuia.

Există o serie de metodologii care pot fi utilizate în construcția modelului, sau a diferitelor tipuri de modelele ale unui sistem de control timp real, ținînd cont de elementele precizate anterior. O sinteză a celor mai cunoscute metodologii pentru dezvoltarea aplicațiilor timp real împreună cu uneltele CASE corespunzătoare va fi prezentată în cadrul capitolului 3.

In plus, indiferent de varianta aleasă, un element esențial îl reprezintă includerea proprietăților temporale în cadrul modelului construit. S-a demonstrat faptul că ***garantarea*** îndeplinirii constrîngerilor de timp impuse de aplicația de control timp real poate fi realizată numai în cazul în care timpii de execuție pentru cazul cel mai defavorabil **WCET** pot fi determinați şi cunoscuți dinainte pentru toate acțiunile cu deadline ferm. In lipsa unor metode analitice de analiză care să poată determina **WCET** ***în orice condiții***, în prezent doar utilizarea controlului temporal **TT** asigură posibilitatea determinării tuturor timpilor **WCET**.

Aceasta datorită faptului că, în cazul acestuia, overhead-ul datorat procesării este constant, perioada de achiziție a task-ului trigger este cunoscută, overhead-ul datorat scanării este de asemenea cunoscut (conform datelor prezentate în Tabelul 1) şi deci **WCET** poate fi determinat; pentru cazul utilizării controlului bazat pe evenimente **ET**, datorită faptului că atît timpul de răspuns la o întrerupere precum şi overhead-ul datorat procesării sunt variabili, determinarea lui **WCET** în aceste condiții nu este posibilă întotdeauna.

Un exemplu de determinare al timpului **WCET** în cazul implementării cu ajutorul PLC-urilor a aplicațiilor de control timp real va fi prezentat în paragrafului 4.2.4. De asemenea, paradigma **Cy-Clone**, descrisă în cadrul paragrafului 4.2., şi care se bazează pe principiul ***resource adequacy*** utilizînd controlul temporal **TT**, se află la baza metodologiei de dezvoltare a aplicațiilor de control timp real propuse în cadrul capitolului 5.

3. METODOLOGII GENERALE DE DEZVOLTARE A APLICAȚIILOR TIMP REAL

3.1. Aspecte esențiale în cadrul metodologiilor de proiectare

Dezvoltarea aplicațiilor complexe de control timp real constituie o problemă dificilă. Sistemele tradiționale au fost proiectate utilizînd de regulă tehnici ad-hoc şi au fost în majoritatea cazurilor, codate utilizînd limbaje de asamblare. Proiectarea acestor aplicații s-a bazat puternic pe primitivele pentru operațiile concurente care au fost puse la dispoziție de sistemul de operare. Chiar dacă au existat anumite metodologii utilizate în acest domeniu, majoritatea acestora considerau limbajul ales şi structura hardware a sistemului ca fiind elemente esențiale care trebuiau luate în calcul în procesul de descompunere a sistemului în task-uri. De asemenea, verificarea formală a sistemului rezultat astfel era în cele mai multe cazuri ignorată, în primul rînd datorită complexității acesteia [How93].

In prezent, programarea orientată pe obiecte oferă multe avantaje în cazul sistemelor de control timp real complexe, inclusiv abilitatea de a reutiliza componentele software existente. Pe de altă parte însă, executabilele object-oriented existente sunt în general de dimensiuni mari şi lente. Acest lucru a făcut pînă în prezent ca programarea orientată pe obiecte să fie mai puțin atractivă pentru proiectanții de sisteme de control în timp real. In prezent însă, au fost dezvoltate noi unelte software care furnizează executive rapide şi care permit construirea de aplicații pe baza componentelor furnizate de acestea şi/sau dezvoltate de utilizator. De asemenea, maşinile virtuale existente în prezent vor putea fi utilizate pentru a furniza independența de platformă a executabilelor object-oriented [JCJG92].

O altă capabilitate importantă prezentă în cadrul uneltelor şi mediilor actuale este posibilitatea de a modela cu ajutorul acestora atît hardware-ul sistemului înglobat (sau părți ale acestuia) cît şi mediul în care sistemul este intenționat să funcționeze. Această modelare dă posibilitatea inginerului de a experimenta, împreună cu aplicația software, comportarea în ansamblu a sistemului încă din faza de proiectare a acetuia, şi chiar în situații în care hardware-ul final nu este încă disponibil. De asemenea, astfel se poate observa dinamica sistemului într-o mare varietate de situații simulate; aceasta reprezintă o facilitate foarte importantă în cazul sistemelor de timp real critice, în cadrul cărora o testare "pe viu" a sistemului poate fi extrem de dificil de realizat şi, în plus, mare consumatoare de timp.

Principalele elemente care apar în procesul de proiectare a sistemelor timp real sunt diferite de alte sisteme în primul rînd datorită prezenței evenimentelor concurente în cerințele temporale care există între intrări şi ieşiri. In acest context, cele mai importante aspecte care trebuie luate în considerare sunt ***descompunerea***, ***interfațarea*** precum şi ***aspectele temporale*** [LA93][Lap93]:

- ***descompunerea*** - deoarece sistemele timp real operează în prezența unor evenimente concurente, este natural să se proiecteze software-ul aferent ca un set de procese concurente. Descompunerea adecvată nu numai că ajută la o mai bună managerizare a complexității sistemului, dar poate fi extrem de importantă de asemenea în asigurarea predictabilității cerințelor temporale.
- ***interfațarea*** - interfața între componentele unui sistem, inclusiv senzori şi actuatori reprezintă forma de comunicare şi sincronizare. Sincronizarea conduce la o mai mare predictabilitate şi o mai uşoară detectare a condițiilor de eorare; pe de altă parte, întîrzierile introduse de mecanismul de sincronizare pot conduce la degradarea nedorită a performanțelor sistemului.
- ***aspectele temporale*** - principalul element în proiectarea sistemelor timp real îl reprezintă modalitatea în care sunt reflectate constrîngerile de timp; astfel, descompunerea sistemului trebuie să se bazeze atît pe cerințe temporale cît şi funcționale. In plus, comportarea temporală a componentelor trebuie analizată din punct de vedere al completitudinii (dacă toate cerințele sunt reflectate în modelul proiectat), corectitudinii (dacă sunt conflicte şi inconsistențe în model) şi al fezabilității (dacă sistemul poate atinge cerințele, adică este fezabil). **Timeout**-ul şi întreruperile de timp sunt formele de bază de monitorizare a comportării temporale a sistemului. Selecția semnalelor de timeout adcvate este foarte importantă: alegerea intervalelor de timeout joacă un rol important în sincronizarea activităților concurente.

Alte aspecte importante, nu neapărat specifice sistemelor în timp real includ:

- ***suport automat pentru proiectare***, din faza de specificare a cerințelor pînă la implementare; în plus, proiectarea unui sistem software complex poate beneficia enorm de unelte de dezvoltare şi depanare interactive, îndeosebi de cele care dispun de o interfață grafică
- ***mijloace pentru reflectarea arhitecturii hardware de-a lungul întregul ciclu de dezvoltare software***: analiza fiabilității şi a performanțelor temporale sunt semnificative numai în măsura în care caracteristicile procesorului (procesoarelor) sunt luate în considerare

3.2. Metodologii de proiectare

Scopul fundamental al tuturor proiectanților de sisteme timp real este acela de a produce un sistem software de calitate, care să satisfacă nu numai cerințele funcționale şi de performanță dar să fie de asemenea fiabil şi uşor de întreținut. Alte cerințe adiționale sunt: sistemul să fie produs la un cost rezonabil şi să fie adaptabil la schimbări, atît software cît şi eventual hardware. In general, o bună proiectare implică o bună practică inginerească, o bună metodologie precum şi existența unor unelte CASE corespunzătoare care să ajute de-a lungul întregului proces de proiectare.

3.2.1. Metode de descompunere a sistemelor (modelarea statică)

Sistemele timp real sunt în mod tipic proiectate ca un set de procese concurente care comunică între ele; din acest motiv, modul de descompunere a sistemului timp real în procese componente reprezintă un element esențial.

3.2.1.1. Proiectarea structurată şi descompunerea funcțională

Această abordare este similară cu metodele de descompunere structurate tradiționale: un sistem este descompus în subsisteme, la rîndul lor aceste subsisteme sunt descompuse în task-uri, etc [Pag88]. Structurarea aplicației în subsisteme se bazează pe descompunerea funcțională în scopul de a obține coeziune mare în cadrul unui subsistem şi coeziune slabă între subsisteme.

Coeziunea funcțională se referă la similaritatea funcțiilor realizate: acest lucru poate necesita schimburi masive de date şi în consecință gruparea în cadrul unui subsistem limitează overhead-ul asociat comunicării. De asemenea, activitățile mai pot fi grupate şi în alte moduri: activități generate de acelaşi eveniment, activități care trebuie să se execute secvențial, etc. Gradul de cuplare între subsisteme determină gradul de concurență şi de întîrzieri datorate sincronizării. Este de dorit în multe cazuri să se micşoreze gradul de cuplare pentru a creşte gradul de concurență asincronă în sistem.

Subsistemele sunt descompuse în continuare în task-uri concurente: cîteva din criteriile de descompunere sunt următoarele [Coo97]:

- ***dependența de I/O*** - din moment ce transformările care depind de intrări şi ieşiri trebuie să ruleze la viteza dispozitivelor de I/O, task-uri separate trebuie utilizate pentru astfel de transformări. Este de dorit să existe cîte un task separat pentru fiecare tip de dispozitiv de I/O asincron

- ***dependența de interfața utilizator*** - ca şi în cazul transformărilor care depind de I/O, transformările care depind de interacțiunile cu utilizatorul trebuie structurate în task-uri separate asincrone
- ***execuții periodice*** - transformările care sunt executate regulat la intervale predeterminate de timp trebuie proiectate ca task-uri separate în scopul de a facilita planificarea. De asemenea, pentru activități de I/O periodice trebuiesc utilizate task-uri independente
- ***funcții cu cerințe temporale critice*** - activitățile care au timp de răspuns critic trebuie de asemenea proiectate sub forma de task-uri separate pentru a permite planificarea cu prioritate a acestora

3.2.1.2. Proiectarea orientată pe obiecte

Metodologiile **object-oriented** diferă de abordările funcționale în primul rînd datorită manierei în care subsistemele, task-urile şi modulele sunt identificate. In cazul metodelor funcționale, fiecare modul reprezintă o transformare majoră. In cazul metodelor orientate pe obiecte, fiecare modul este responsabil de managerizarea unui obiect major din sistem [IBR98].

Cu alte cuvinte, în varianta funcțională mai multe module pot opera asupra unui obiect realizînd diferite transformări ale acestuia; în varianta ***object-oriented***, unui singur modul îi este permis să opereze asupra unui obiect, realizînd toate transformările asupra acestuia.

Diferiți paşi sunt utilizați de-a lungul dezvoltării orientate pe obiecte. Primul pas îl reprezintă identificarea obiectelor şi a atributelor acestora. In mod tipic aceasta implică identificarea principalelor obiecte de tip ***actor***, ***agent*** şi ***server*** în spațiul problemei date şi definirea rolului acestora în sistem [Boo98]. Următorul pas îl reprezintă identificarea transformărilor care trebuiesc efectuate asupra obiectelor şi a serviciilor, dacă există, pe care trebuie să le ofere fiecare obiect identificat. Pentru a satisface principiul ascunderii informației, este de asemenea necesară definirea interfețelor vizibile ale fiecărui obiect. In final, serviciile şi transformările asociate cu fiecare obiect sunt implementate; evident, detaliile de implementare nu trebuie să fie vizibile către alte obiecte.

Atunci cînd se utilizează o metodologie orientată pe obiecte, se pot utiliza transformările şi serviciile comune unei categorii de obiecte (în fazele inițiale ale proiectării) pentru a defini clase de obiecte şi ierarhii de clase; acesta poate conduce la utilizarea proprietăților de moştenire şi polimorfism pentru definirea sub-claselor şi a transformărilor asupra diferitelor tipuri de date [Eli95].

Abordarea obiectuală pare naturală pentru sistemele timp real, datorită capabilității acesteia de abstractizare obținută prin mecanismele de ascundere a informației, moştenire şi polimorfism; în plus, reutilizarea şi mentenabilitatea codului reprezintă un atribut pozitiv [Kru92].

Totuşi, abordarea are şi cîteva nejunsuri destul de importante. In primul rînd, sistemul conține de cele mai multe ori prea multe obiecte. In al doilea rînd, şi cel mai important, este mult mai greu de anticipat comportarea din punct de vedere temporal față de varianta funcțională. Acest lucru se întîmplă datorită faptului că transformările din cadrul sistemului, cu care sunt asociate constrîngeri de timp, pot consta din fire de execuție multiple între mai multe obiecte.

3.2.1.3. Maşini virtuale

Din punct de vedere ierarhic, maşinile virtuale pot fi utilizate pentru proiectarea sistemelor timp real complexe. In mod tipic, acest lucru implică crearea unui set de abstracții pe niveluri care simplifică procesul de proiectare, lăsînd detaliile de implementare pentru nivelul cel mai de jos, asociat maşinii propriu-zise.

Astfel, pe nivelurile înalte de abstractizare, reprezentarea maşinii virtuale a unui sistem de timp real constă în mod normal din procese, canale, buffer-e, date utilizate în comun, evenimente de sincronizare şi întreruperi. Procesele reprezintă agenții care implementează activitățile şi funcțiile. Procesele comunică unele cu altele utilizînd canale şi buffer-e, iar sincronizarea este realizată prin evenimente.

Canalele conectează două procese şi pot fi uni- sau bi-direcționale. Buffer-ele asociate cu canalele permit comunicarea asincronă. Utilizarea datelor comune este folosită pentru reprezentarea comunicării între diferite procese (one to many sau many to many). Wait, Signal şi Interrupt sunt printre cele mai utilizate tipuri de evenimente de sincronizare. Descompunerea sistemelor în acest caz poate fi bazată atît pe varianta funcțională cît şi pe varianta object-oriented.

In mod teoretic, dacă este bine utilizată, această abordare promovează un înalt grad de mentenabilitate şi portabilitate, cîtă vreme este posibilă înlocuirea oricărui nivel cu un nivel nou. Din punct de vedere practic însă, în majoritatea sistemelor acest lucru nu este atît de uşor de realizat datorită dependențelor între abstractizările existente pe diferitele niveluri.

3.2.1.4. Metode *entity-life*

Există şi alte metode de descompunere care pot fi utilizate pentru sistemele timp real. Una dintre acestea este aşa numitul model ***entity-life***. Este destul de dificil de definit foarte exact ce presupune o astfel de modelare: în principiu, criteriile de structurare a modelului (definirea proceselor concurente, task-uri, comunicare şi sincronizare) se bazează pe entitățile şi construcțiile furnizate de limbajul de implementare [Sad89].

După cum este ilustrat prin metodele de descopunere prezentate mai sus, nu există reguli stricte care să guverneze descompunerea; în majoritatea cazurilor o bună descompunere rezultă dintr-o bună înțelegere a cerințelor, din utilizarea unui limbaj de programare de nivel înalt şi este funcție de arhitectura hardware aleasă pentru implementare. O aderență prea strictă la oricare din metodologiile de descompunere existente poate conduce la proiectări care sunt fie ineficiente fie nenaturale. Din acest motiv, mediile CASE de proiectare software şi uneltele lor asociate trebuie să permită utilizarea a multiple metodologii.

3.2.2. Metode de abstractizare ale proceselor (modelarea dinamică)

Odată subsistemele (obiecte, task-uri, module) au fost identificate, activitățile realizate în cadrul fiecărui subsistem şi interfețele între acestea trebuie definite. Task-urile concurente necesită comunicarea între ele pentru schimbul de date, control şi utilizarea în comun a datelor. Este comună utilizarea de abstracții grafice pentru reprezentarea operațiilor din cadrul subsistemelor: cele mai larg răspîndite sunt ***diagramele de stări finite***, ***diagramele de flux*** şi ***rețelele Petri*** [Gom93].

3.2.2.1. Diagrame de stări finite (Finite State Diagram FSD)

Diagramele de stări finite FSD sunt utilizate frecvent pentru a descrie comportarea proceselor prin listarea tuturor stărilor posibile al acestora, a intrărilor care au ca efect trecerea procesul dintr-o stare în alta precum şi a ieşirilor produse de proces în fiecare stare. Diagramele de stări finite sunt în mod normal reprezentate grafic prin intermediul diagramelor de stări. Atunci cînd se utilizează diagrame de stări finite ca şi modalitate de abstractizare, fiecare stare reprezintă completarea unei activități. Funcțiile realizate de procese sunt abstractizate în stări, implicînd faptul că procesul realizează operațiile de transformare cît timp rezidă într-o stare. Procesul acceptă intrări şi tranziții către starea următoare numai după completarea activităților din starea curentă.

3.2.2.2. Diagrame de flux extinse

O diagramă de flux pentru un proces constă din trei elemente:

- transformări de date: funcțiile realizate de proces, în mod normal reprezentate prin cercuri sau pătrate
- săgeți între noduri prin care se ilustrează modul de curgere a datelor între transformări
- înmagazinări de date (data stores)

Chiar dacă diagramele de flux au fost utilizate extensiv, acestea nu sunt potrivite pentru reprezentarea proceselor timp real. De exemplu, acestea nu pot descrie dependența transformărilor de timp. Pentru a preîntîmpina aceste limitări, au fost propuse cîteva extensii al diagramelor de flux inițiale. In principiu, în cadrul acestora două tipuri de transformări există: de date şi de control. Transformările de control definesc modul în care transformările de date sunt activate sau inactivate. Aceste transformări de control trebuie asociate cu diagrame de stări finite pentru a reprezenta secvența de acțiuni care trebuie realizată de un proces. Chiar şi în aceste condiții însă, datorită utilizării diagramelor de stări finite, va rezulta în final un model neuniform.

3.2.2.3. Rețele Petri

Rețelele Petri au cîştgat o popularitate considerabilă de-a lungul ultimei decade datorită abordării grafice şi a abilității acestora de a reprezenta concurența asincronă. Rețelele Petri constă din două feluri de noduri: ***locuri*** **(places)** şi ***tranziții*** **(transitions)**. Tranzițiile reprezintă activități, funcții sau transformări, în timp ce locurile reprezintă evenimente ca de exemplu semnale sau completarea unor activități. Există la ora actuală diferite extensii ale rețelelor Petri, ca de exemplu rețele **Timed Petri**, care sunt mai potrivite pentru sistemele timp real decît cele orignale.

O altă modalitate de abstractizare, cu un mecanism similar cu cel al rețelelor Petri, îl reprezintă ***maşina ierarhică cu mai multe stări*** **(Hierarchical Multistate Machine HMS)**. Ideea este aceea de a păstra concurența chiar în cazul utilizării unor abstracții de nivel înalt. Maşina ierarhică cu mai multe stări permite definirea unor tranziții colective a unui set de task-uri, în încercarea de a reduce creşterea exponențială ca număr a posibilelor stări, definind în loc de obiecte, mulțimi. Obiectele reprezintă valori ale datelor. Acest lucru este în contrast cu abstractizările care definesc stări separate pentru fiecare valoare posibilă a datelor.

In general, abstractizările utilizate pentru reprezentarea proceselor sunt fie sincrone fie asincrone, prin natura lor. Modelele bazate pe diagramele de stări finite sunt sincrone. In cadrul acestora, in multe situații, specificațiile de timp

real pot fi realizate numai dacă se tratează timpul ca şi o varabilă de stare mai degrabă decît o funcție a unei tranziții între stări. Modelele bazate pe rețele Petri şi diagrame de curgere sunt asincrone. Evenimentele care generează o acțiune sunt mult mai uşor de modelat utilizînd astfel de abstracții.

3.2.3. Metodologii de implementare

In faza finală a proiectării sistemului timp real, reprezentarea abstractă a proceselor şi a interfețelor acestora este translatată în programe într-un limbaj, de programare ales. Acest lucru necesită proiectarea structurilor de date, a structurilor de control, a structurii task-urilor precum şi a modalității de comunicare şi sincronizare între task-uri. Evident, alegerea este constrînsă de limbajul de programare ales.

Atunci cînd limbajul nu furnizează mecanisme pentru definirea şi controlul task-urilor concurente, proiectantul este nevoit să se bazeze pe primitivele puse la dispoziție de sistemul de operare. Este foarte important să se înțeleagă bine modelul de concurență şi comunicare pus la dispoziție de limbaj pentru a obține o execuție corectă şi predictibilă a sistemului [You98].

De exemplu, mecanismul activării task-urilor este diferit în diferite limbaje. Task-urile pot fi active tot timpul, pot deveni active atunci cînd subrutina care le conține este invocată, sau pot fi activate la cerere. Activarea task-urilor la momente specifice de timp este suportată în unele limbaje. Totuşi, în majoritatea cazurilor programatorul trebuie să ajusteze specificațiile de timp pentru activarea task-urilor pentru a satisface cerințele aplicației. Astfel, flexibilitatea creşte dacă se utilizează legarea dinamică (dynamic binding) însă creşte şi regia (overhead-ul) în execuție. Unele limbaje permit definirea modului de tratare a excepțiilor: evenimente deosebite pot fi tratate utilizînd această facilitate.

Principiul ascunderii informației poate fi implementat uşor dacă limbajul furnizează tipuri de date abstracte, module, etc. Astfel de abstracții permit definirea interfețelor vizibile ascunzînd implementarea; limbajele orientate pe obiecte [ITM90], prin mecanismul de încapsulare, facilitează ascunderea informației.

Deseori, sistemele înglobate trebuie să trateze I/O de nivel scăzut provenite de la senzori şi alte tipuri de instrumente. Acestea sunt dependente de maşină şi în consecință multe limbaje ignoră aspectele legate de I/O, bazîndu-se pe suportul oferit de sistemul de operare. Alte limbaje, mai recente, specifice pentru sistemele înglobate, sunt proiectate însă special pentru a putea trata cu uşurință problema I/O.

3.3. Unelte şi medii de proiectare şi modelare

Aplicarea metodologiilor existente nu ar fi posibilă fără existența unor unelte CASE adecvate care să asiste programatorul de-a lungul întregului proces de proiectare a aplicatiei timp real, pe cît posibil automatizînd anumite etape. Astfel, un mediu integrat de dezvoltare trebuie să permită programatorului să modeleze, de preferință în mod grafic fiecare aspect din sistem, atît din punct de vedere static cît şi dinamic.

In cadrul uneltelor şi mediilor de proiectare existente în prezent, interfața de modelare este bazată actualmente pe standardele existente, dintre care cele mai utilizate sunt:

- ***JSC (Jackson Structured Charts)*** care utilizează descompunerea funcțională ca şi tehnică de descompunere; modelul abstract **JSC** constă dintr-o rețea de procese, de intrare, de ieşire şi interne, conectate prin intermediul buffer-elor asincrone sau utilizînd vectori de stare [Coo97]. Diferite variante de astfel de diagrame structurate se regăsesc în uneltele de proiectare existente la ora actuală

- ***OMT (Object Modeling Technique)*** şi ***UML (Unified Modeling Language)***, care permit programatorului să descrie sistemul utilizînd abordarea orientată pe obiecte. Notația OMT sau UML este utilizată pentru a construi modelul obiect al sistemului pornind de la specificații. Sunt utilizate în acest sens conceptele binecunoscute de clasă, atribute, relații şi metode. Descrierea claselor împreună cu relațiile dintre ele formează aşa numita diagramă de clase a aplicației respective [Bri98a][Bri98b]

- ***SDL (Specification and Description Language)*** definit de International Telecommunications Union: Tele-communications body (ITU-T), care permite programatorului să construiască diagrame de stări finite care să înglobeze comportarea dorită a componentelor sistemului. Acesta este utilizat în faza de proiectare de ansamblu şi se bazează pe conceptele de sistem, block şi proces. Arhitectura sistemului este descrisă prin intermediul unei diagrame SDL ierarhice, formată dintr-un set de diagrame SDL interconectate. Elementele din cadrul arhitecturii definite comunică între ele prin semnale transportate prin intermediul unor canale. Fiecare proces SDL este descris prin intermediul diagramelor de stări finite, în faza de proiectare detaliată [Ver96]

- ***MSC (Message Sequence Charts)***, care permite programatorului să descrie secvențele de interacțiune între diferitele componente ale sistemului. Fiecare diagramă MSC ilustrează un singur caz de schimb de informații între componentele sistemului şi diferiți actori din cadrul mediului exterior [Ver96]

Alte facilități suplimentare de care un mediu integrat ar trebui să dispună ar fi următoarele:

- abilitatea de a produce sisteme de calitate prin aplicarea tehnicilor de verificare şi validare cunoscute începînd primele faze din cadrul ciclului de dezvoltare
- posibilitatea de a scurta timpul de dezvoltare prin utilizarea modelelor grafice obținute în scopul generării automate a codului sursă, care ulterior să poată fi compilat şi link-editat pentru a rula pe o mare diversitate de sisteme
- posibilitatea de a permite anticiparea tuturor schimbărilor posibile încă din faza de proiectare, menținînd astfel intervenția în codul sursă la minimum şi încurajînd reutilizarea codului.

In consecință, în măsura în care este posibil, mediul integrat trebuie să poată automatiza şi controla cît mai mult posibil din procesul de dezvoltare, începînd cu analiza cerințelor, generarea codului sursă, testarea şi chiar întreținerea.

Cîteva dintre cele mai cunoscute şi utilizate unelte CASE şi medii de proiectare şi modelare împreună cu cele mai importante caracteristici ale acestora vor fi prezentate în continuare.

3.3.1. MGA ACSL/Graphic Modeller

Produsul **Graphic Modeller** al firmei MGA reprezintă o unealtă CASE care poate fi utilizată pentru modelarea sistemelor de control timp real. Ca şi metodă de descompunere, **Graphic Modeller** utilizează descompunerea funcțională în subsisteme, blocuri şi sub-blocuri, în varianta **top-down**. Modelele descrise cu ajutorul mediului **Graphic Modeller** pot fi ulterior simulate utilizînd limbajul de simulare **ACSL** [MGA96a] [MGA96b].

Limbajul de simulare continuă **ACSL (Advanced Continuous Simulation Language)** a fost dezvoltat special cu scopul de a modela sisteme descrise funcție de dependența de timp, prin ecuații diferențiale neliniare şi/sau funcții de transfer [MGA95]. Pentru **ACSL**, modelul trebuie exprimat în termeni matematici, ecuațiile diferențiale neliniare fiind des utilizate.

De asemenea, unele părți pot fi exprimate şi prin funcții de transfer din teoria sistemelor liniare; pentru sistemele reale, în afară de relațiile cauză/efect, mai

sunt necesare şi includerea unor specificaţii referitoare la limitări, discontinuităţi, precum şi alte condiţii care limitează domeniul de funcţionare al sistemului.

In continuare vor fi evidenţiate cele mai importante elemente ale mediului **ACSL/Graphic Modeller**, împreună cu o scurtă descriere a capabilităţilor şi caracteristicilor acestora.

3.3.1.1. Editorul grafic Graphic Modeller

După cum a fost precizat anterior, produsul **Graphic Modeller** reprezintă un mediu de execuţie pentru analiza modelelor **ACSL**, care conţine inclus şi un editor grafic, deosebit de flexibil şi uşor de utilizat pentru crearea modelelor într-o formă grafică. Cu ajutorul interfeţei extrem de prietenoase, programatorul poate vizualiza, crea şi modifica toate elementele modelului. In plus, utilizînd software-ul **Graphic Modeller** se mai pot realiza următoarele:

- analiza şi proiectarea sistemelor în termeni de diagrame bloc utilizînd componente predefinite sau definite de utilizator; astfel se pot crea bibioteci de astfel de componente reutilizabile (sistem modular)
- testarea unei singure componente a modelului la un anumit dat: acest lucru facilitează practic crearea de modele complexe. In acest sens editorul Graphic Modeller reprezintînd un participant activ în cadrul procesului de dezvoltare a modelului

Modelele realizate astfel sunt păstrate în fişiere .gm. Atunci cînd un astfel de model este executat, **Graphic Modeller** generează automat cod **ASCL** (fişier .csl) care în continuare este translatat în Fortran (.for), compilat şi linkeditat rezultînd în final fişierul executabil .prx.

3.3.1.2. Simulatorul ACSL

Software-ul **ACSL** furnizează metode simple de a reprezenta modelele matematice pe un calculator (PC). Practic, există două modalităţi de lucru cu software-ul **ASCL**:

- utilizînd limbajul **ASCL**, pe baza ecuaţilor matematice, utilizatorul scrie un program care modelează sistemul (fişier .csl); programul este apoi translatat de **ASCL** într-un fişier Fortran (.for), care apoi este compilat (.obj) şi link-editat utilizînd bibliotecile **ACSL** şi Fortran necesare, rezultînd un fişier executabil (.prx)
- utilizînd opţiunea **Graphic Modeller** care pratic furnizează în plus o interfaţă grafică pentru crearea modelului, şi care poate fi translatată automat în cod **ASCL**, după care procesul descris anterior se repetă

Execuția modelului obținut poate fi ulterior realizată prin comenzi furnizate în loturi (batch), sub forma unor fişiere-scenariu predefinite, sau interactiv, una cîte una. În varianta interactivă, se poate experimenta mai flexibil comportarea sistemului schimbînd diferite constante şi urmărind comportarea acestuia.

Din punct de vedere al limbajului de simulare **ACSL**, majoritatea instrucțiunilor recunoscute de translatorul **ACSL** sunt echivalente cu instrucțiunile Fortran corespunzătoare, cu excepția faptului că **ACSL** nu are nici un fel de restricții referitoare la plasamentul pe anumite coloane a codului. Ca orice limbaj de programare, **ACSL** conține instrucțiuni (operatori) pentru:

- ***realizarea diferitelor operații matematice*** - fiind un limbaj de simulare care se bazează în mod fundamental pe descrierea prin ecuații matematice a sistemelor, **ACSL** trebuie să furnizeze un puternic set de instrucțiuni şi operatori în acest scop, dintre care amintim: operatori booleeni, operatori de conversie în timp real (conversie analog-digital), operatori aritmetici, operatori trigonometrici (funcții trigonometrice, hiperbolice), operatori de integrare INTEG, etc.
- ***vizualizarea pe diferite tipuri de grafice*** – prin intermediul cărora se poate urmări evoluția diferitelor mărimi în cadrul procesului de simulare
- ***structurarea şi controlul programului*** – sunt utilizate în acest scop instrucțiuni de tipul IF, IF-THEN-ELSE, DO, precum şi instrucțiuni de asignare, declarare a diferitelor variabile şi constante, etc.
- ***lucrul cu fişiere*** – pentru stocarea datelor obținute în urma simulării
- ***alte instrucțiuni şi operatori specifici*** – din această categorie fac parte operatori tip regulator generic (P, PI, PD, PID), operatori neliniari: (comutator logic, comutator logic, zonă moartă, histereză, limitare, discretizare, etc.), operatori liniari (întârziere, amplificare, funcții de transfer generice de ordinul I şi II, etc.), operatori sursă (treaptă, rampă, constantă, puls, etc.), operatori pentru generare numere aleatoare (uniform distribuite,distribuție Gauss), etc.

Practic însă, operatorii de integrare reprezintă inima limbajului de simulare **ACSL**. Chiar dacă limbajul de simulare realizează integrarea, utilizatorul este cel care trebuie să precizeze algoritmul de integrare utilizat precum şi mărimea pasului de integrare. **ACSL** pune la dispoziție mai multe tipuri de algoritmi de integrare: Adams-Moulton, Gear's Stiff, Runge-Kutta, Runge-Kutta-Fehlberg, etc. Cel mai recomandat algoritm de integrare utilizat pentru sisteme de control timp real este algoritmul Runge-Kutta de ordinul doi, deoarece este cel mai stabil şi mai exact.

Astfel, pentru calculul variației în timp a valorii unei componente, limbajul de simulare realizează integrarea expresiei derivatei în raport cu timpul, începînd cu o stare inițială cunoscută. Variabila timp este este considerată în modelul matematic variabila independentă. Valoarea sa inițială este zero, şi aceasta este incrementată cu o valoare egală cu pasul de integrare ales. Eficiența unei simulări este direct influențată de complexitatea sistemului precum şi valoarea pasului de integrare.

Astfel, structura generală a unui program sursă **ACSL** are următoarea formă :

```
PROGRAM
   INITIAL
     instrucţiuni care sunt executate
       înainte de începerea rulării programului
     variabilele de stare nu conţin încă condiţiile iniţiale
   END
   DYNAMIC
        DERIVATIVE
          instrucţiuni de integrare continuă
        END
        DISCRETE
          instrucţiuni executate la momente de timp
          specificate
        END
     instrucţiuni executate la fiecare interval
    de comunicare
   END
   TERMINAL
     instrucţiuni executate după rularea programului
   END
END
```

Astfel, se observă că programul începe secvențial cu secțiunea **INITIAL**, secțiune în care se fac calculele dinainte de rulare, respectiv variabilele de stare sunt inițializate cu valorile corespunzătoare sau sunt calculate unele condiții inițiale; practic orice variabilă care nu îşi modifică valoarea în timpul rulării poate fi calculată în această secțiune.

Rutina de integrare este inițializată când se face transferul controlului la ieşirea din secțiunea **INITIAL**. Această inițializare presupune transferarea tuturor valorilor condițiilor inițiale variabilelor şi evaluarea codului în secțiunile **DERIVATIVE** şi **DISCRETE**. Acest lucru se realizează pentru a fi sigur că toate variabilele calculate prin program sunt cunoscute înainte de înregistrarea primelor date rezultate din execuție.

După inițializarea şi evaluarea codului din secțiunile **DERIVATIVE** şi **DISCRETE**, fanionul **STOP** este resetat şi programul execută instrucțiunile din cadrul secțiunii **DYNAMIC**.

După evaluarea secțiunii **DYNAMIC**, este verificat fanionul **STOP**; dacă este setat se va transfera controlul secțiunii **TERMINAL**. Dacă fanionul **STOP** nu a fost setat, programul afişează valorile variabilelor specificate prin instrucțiunile **OUTPUT** sau **PREPARE** pe ecran sau în fişiere de unde ulterior pot fi afişate pe ecran.

Rutina de integrare va face integrarea pe o perioadă egală cu intervalul de comunicare (sau până la întâlnirea unei condiții de terminare) folosind instrucțiunile din secțiunea **DERIVATIVE** pentru evaluarea derivatelor variabilelor de stare. După fiecare interval de comunicare este testat fanionul **STOP**. Dacă acesta e setat se va trece la execuție secțiune **TERMINAL** , în caz contrar se reia execuția secțiunii **DYNAMIC**.

Evoluția sistemului poate fi urmărită prin intermediul evaluării unor mărimi: simularea ne furnizează o secvență de valori care formează aşa numiții vectori de stare. Într-un sistem dinamic, valorile variabilelor descriu comportarea acestora în timp. Viteza de variație (derivata) fiecărei componente este exprimată în modelul matematic ca şi o combinație algebrică a vectorilor de stare a componentelor (şi eventual derivatele acestora). Ecuațiile trebuie să exprime modul în care viteza de variație a unei componente în timp depinde de celelalte componente precum şi de factorii care acționează din exterior asupra sistemului.

Dacă sistemul ajunge după un anumit timp într-o stare de echilibru, în care derivatele componentelor sunt nule şi nici o variabilă nu-şi mai modifică valoarea, s-a obținut regimul staționar. De exemplu, stabilirea constantelor regulatoarelor este o aplicație în acest sens; acestea sunt stabilite astfel încît sistemul să nu fie oscilant, şi să se stabilizeze în timp. Acest lucru asigură protecție la utilizarea dispozitivelor controlate (valve, motoare) şi este dificil de realizat în timpul funcționării procesului [18]. Dar, după cum a fost precizat anterior, comportarea modelului poate fi analizată urmărind datele afişate pe ecran, sub formă numerică sau, dacă se doreşte se pot trasa grafice care să evidențieze mai clar comportarea diferitelor mărimi, **ACSL** permițind acest lucru.

3.3.1.3. Generare automată de cod

Limbajul **ACSL** dispune de opțiunea **ACSLCode** prin intermediul căreia se generează cod C optimizat pentru aplicațiile de control în timp real. In plus, codul generat poate fi accesat de alte produse din gama **MGA (ACSL, ACSLMath, ACSLVision)**, astfel încît ulterior să poată fi realizate prelucrări şi analize statistice, în afara programului de control.

3.3.2. Real-time Rational ROSE

Rational Rose reprezintă una dintre cele mai utilizate unelte de modelare la ora actuală, şi care are la bază abordarea obiectuală [Rat98]. Ca şi alte unelte de modelare bazate pe tehnologia orientată pe obiecte (de exemplu, **ObjectGeode**, descris ulterior), **Rose** utilizează modelarea vizuală: adică modelarea utilizînd notații grafice standard. In acest caz există trei posibilități: **OMT**, **UML** şi **Booch**, conversia între acestea realizîndu-se automat.

Este posibilă vizualizarea modelului atît din punct de vedere static (cu clase) cît şi dinamic (cu obiecte), cu diferite grade de rafinare, utilizînd diferite tipuri de diagrame. Intrucît diagramele sunt utilizate pentru a vizualiza diferite aspecte ale modelului, icoana reprezentînd un anumit element din model poate apărea într-una sau mai multe astfel de diagrame. Cele mai importante elemente ale uneltei CASE **Rational Rose**, împreună cu o scurtă descriere a capabilităților şi caracteristicilor acestora vor fi evidențiate în continuare.

3.3.2.1. Mecanisme pentru dezvoltare iterativă şi suport multi-utilizator

Utilizînd **Rose**, întregul proces de dezvoltare a aplicației are loc prin intermediul unei secvențe de iterații. Fiecare iterație începe cu aprecierea modelului pentru a identifica situațiile critice, nerezolvate. Sunt identificate scenarii ilustrînd riscurile şi modelul curent este extins pentru a adresa aceste situații. Pentru a furniza dovada că aceste riscuri au fost luate în considerare, implementarea curentă este extinsă şi sunt construite teste pentru scenariile mai sus introduse. Este posibil ca şi modelul să evolueze după ce o anumită variantă de implementare a fost construită; în acest caz, actualizarea se va realiza de ambele părți. Cînd aceasta este definitivă, începe următoarea iterație.

Dezvoltarea iterativă este practicată în general de echipe de ingineri software care operează în paralel, în mod tipic utilizînd subsisteme din cadrul modelului pentru a partiționa munca. Pentru a se putea realiza o dezvoltare în paralel eficientă, este necesar ca fiecare să aibă propriul spațiu de lucru conținînd însă toate componentele din model.

Suportul pentru programare multi-utilizator furnizat de **Rose** furnizează fiecărui programator un **workspace** privat şi integrat cu ajutorul componentei numită **Configuration Management** pentru a suporta propagarea controlată a schimbărilor şi menținerea integrității fiecărei categorie de clase şi subsisteme în parte. Prin memorarea fişierelor asociate modelului pe un suport de memorare extern, **Rose** dă posibilitatea memorării diferitelor versiuni ale modelului şi de creare a unor arhive care pot fi ulterior regăsite şi eventual reutilizate.

3.3.2.2. Editorul grafic Rational Rose

Avînd în vedere scopul de a găsi erori cît mai devreme în decursul ciclului de dezvoltare al unei aplicații, o importantă caracteristică a mediului de dezvoltare o reprezintă editorul grafic. Acesta trebuie să fie un participant activ în cadrul procesului de dezvoltare: trebuie să detecteze erorile de notație din mers şi să înştiințeze programatorul; să țină cont de contextul existent şi să permită crearea numai a construcțiilor consistente din punct de vedere al limbajului utilizat.

In plus, editorul **Rose** permite programatorului să vizualizeze, creeze, să modifice şi să manipuleze toate elementele modelului. Interfața intuitivă furnizează palete, meniuri, funcții standard de tipul **drag and drop**, facilități de **undo/redo**, etc. Deoarece sunt posibile diferite vederi asupra unui model, schimbările într-una sunt reflectate automat în toate celelalte. Editorul dispune de asemenea de un **dicționar** al tuturor obiectelor curente şi care menține legătura dinamică între obiectele asociate, în scopul de a localiza uşor obiectele corespondente în fiecare din reprezentările grafice posibile.

Modelul general al applicației este descris utilizînd diferite elemente: clase, scenarii (**use-cases)**, obiecte, pachete logice (**logic packages)**, operații, componente, pachete de componente (**component packages)**, procesoare, etc. împreună cu relațiile dintre ele. Fiecare element al modelului îşi are propriile proprietăți care îl identifică şi îl caracterizează. Notația folosită furnizează icoane grafice pentru a reprezenta fiecare tip de element şi relație.

3.3.2.3. Facilități de *inginerie inversă*

Ingineria inversă reprezintă procesul prin care se examinează codul sursă al unui program în scopul obținerii informațiilor legate de modalitatea de proiectare a acestuia. **Rational Rose/C++ Analyzer** poate extrage informații dintr-o sursă C++ (disponibil de asemenea pentru Java şi Visual Basic) pe care le utilizează pentru a construi modelul reprezentînd structura logică şi fizică a aplicației. **Rose** permite de asemenea vizualizarea şi manipularea ulterioară a acestui model utilizînd notația **UML**, **OMT** sau **Booch**.

3.3.2.4. Generarea automată de cod

O altă caracteristică importantă o reprezintă generarea automată de cod direct din informațiile conținute în model. Codul generat pentru fiecare componentă a modelului este în funcție de specificarea componentei şi de proprietățile modelului.

Proprietățile modelului furnizează informații specifice unui anumit limbaj necesare pentru maparea modelului într-o implementare în limbajul respectiv (C++, Java, Visual Basic).

Practic, scheletul aplicației este generat de **Rose**, bazat pe interfețele definite ale obiectelor, restul implementării este nenesar să se realizeze manual. Acest lucru nu reprezintă neapărat un dezavantaj: astfel pot fi realizate anumite optimizări în cadrul procesului de codificare. Utilizarea generatorului de cod **Rose Code Generator** reduce substanțial timpul dintre proiectare şi execuție, producînd de asemenea fişiere sursă uniform structurate, şi promovînd totodată un stil de codificare consistent şi comentat.

3.3.3. Verilog ObjectGeode

ObjectGeode este o altă unealtă CASE care foloseşte metodologia orientată pe obiecte, şi care este de asemenea dedicată dezvoltării aplicațiilor timp real [Ver96]. în mod asemănător cu Rational Rose, ea constă dintr-un mediu integrat care include un editor grafic pentru crearea diagramelor OMT, MSC şi SLC cu verificatoare statice şi dinamice, un constructor **(builder)** de aplicație care generează cod executabil şi portabil precum şi mecanisme de a asigura consistența la nivel global de-a lungul întregului ciclu de dezvoltare. Cele mai importante elemente ale mediului integrat **ObjectGeode** vor fi prezentate în continuare.

3.3.3.1. Editorul grafic Object Geode

Majoritatea modelelor sunt exprimate într-o formă grafică; structurile grafice sunt verificate din punct de vedere al consistenței. De asemenea, programatorul poate utiliza verificatoarele **(checkers)** corespunzătoare pentru a verifica în orice moment respectarea regulilor **OMT**, **SLC** şi **MSC**, schimbările propagîndu-se automat în toate reprezentările modelului. La fel ca şi **Rose**, **ObjectGeode** furnizează un dicționar complet de obiecte **OMT**, **SDL** şi **MSC**, menținînd legăturile dinamice între obiectele asociate.

3.3.3.2. Generarea automată de cod

Generatorul de cod **OMT++ Code Generator** translatează clasele **OMT** în schelet de cod C++; definirea metodelor poate fi realizată utilizînd editorul **Object Editor**, pentru a genera cod executabil complet. Pe baza modelului **SDL**, generatorul de cod poate produce cod sursă C care apelează serviciile SDL în cadrul unui context **real-time multitasking**.

Pentru cazul în care sistemul de operare timp real nu este proiectat să furnizeze astfel de servicii, ObjectGeode dispune de o ***bibliotecă specială* run-time** care furnizează o maşină virtuală SDL pentru fiecare sistem de operare în timp real. Astfel, generatorul de cod sursă C produce cod care apelează în acest caz funcțiile acestei biblioteci în locul apelurilor unui sistem de operare particular. Acest mod de lucru asigură portabilitatea aplicației pe diferite sisteme de operare în timp real.

Versiunea curentă a bibliotecii **ObjectGeode** este disponibilă în prezent pentru următoarele sisteme de operare: VRTX de la Microtec Research, VxWorks de la Wind River, PSOS+ de la Integrated Systems, OSE de la Enea Data, Chorus de la Chorus Systems precum şi diferite versiuni de UNIX.

După cum a fost precizat anterior, în cazul **ObjectGeode**, se poate genera (prin furnizarea codului metodelor corespunzătoare) automat şi codul complet al aplicației. Datorită acestei automatizări, este posibil ca în anumite situații codul generat să nu îndeplinească toate cerințele aplicației timp real. In scopul îmbunătățirii eficienței, performanța codului C generat poate fi optimizată prin adaptarea bibliotecii **run-time** precum şi ulterioar a codului sursă generat. Aceste mecanisme sunt disponibile în **ObjectGeode** prin intermediul utilizării macro-urilor.

3.3.3.3. Simulatorul ObjectGeode

In procesul de dezvoltare a sistemelor timp real, una din problemele importante care se pun repezintă ajustarea software-ului pe arhitectura finală. In acest scop, **ObjectGeode** furnizează un simulator care permite simularea programului de aplicație înaintea implementării codului sursă. Datorită nivelului de detaliere furnizat de SDL, comportarea sistemului poate fi relativ uşor simulată; de asemenea, utilizînd chart-urile MSC, care permit descrierea unambiguă a comportării dorite de la sistem, acesta poate fi validat în consecință.

In plus, de-a lungul fazei de verificare, simulatorul poate de asemenea detecta dacă modelul SDL este fiabil, semnalînd probleme de genul: erori aritmetice, cod fără efect, depăşiri, **deadlock**-uri, **livelock**-uri.

Există trei moduri în care simulatorul poate funcționa: ***simulare aleatoare* (random simulation mode)**, ***simulare exhaustivă* (exhaustive simulation mode)** şi ***simulare interactivă*** sau ***pas cu pas* (interactive step-by-step simulation mode)**. In cazul simulării aleatoare căile **(behavioral paths)** posibile sunt explorate aleator.

In contrast cu aceasta, modul de simulare exhaustivă necesită explorarea tuturor căilor posibile. Atunci cînd rulează în acest mod, simulatorul trebuie să verifice toate proprietățile necesare validării. Pentru a fi eficient, acest mod de simulare necesită însă performanțe extrem de ridicate pentru maşina pe care rulează, datorită numărului mare de căi care trebuiesc explorate. Dar, dacă simularea exhaustivă se termină bine, aceasta poate dovedi două dintre cele mai importante proprietăți ale sistemului în timp real: ***siguranța în funcționare* (safety)**, adică faptul că sistemul nu reacționează niciodată altfel decît este prevăzut, precum şi proprietatea de **liveness**, adică aceea că toate comportările posibile ale sistemului au fost prevăzute.

3.4. Concluzii

In concluzie, următoarele elemente trebuie luate în considerare atunci cînd se proiectează o aplicație timp real:

- ***diversitatea sistemelor timp real*** – chiar dacă ele au anumite aspecte comune, acestea pot diferi foarte mult în ce priveşte dimensiunea, complexitatea, cît sunt de critice sau în ce priveşte mediul de execuție pe care rulează. Unele dintre acestea constă numai din cîteva mii de linii de cod, în timp ce altele pot avea milioane de astfel de linii. Unele pot necesita verificarea formală din punct de vedere logic şi al cerințelor temporale pentru a fi ultrafiabile, altele nu sunt atît de critice. Deci, este important pentru fiecare sistem timp real particular să se identifice cea mai potrivită metodă de proiectare şi cele mai potrivite strategii de verificare şi validare
- ***specificarea şi analiza timpului*** – teoretic, analiza cerințelor temporale a unui sistem de timp real trebuie realizată de-a lungul diferitelor faze din cadrul procesului de dezvoltare, inclusiv specificarea cerințelor, proiectarea şi implementarea, din punct de vedere al consistenței, completării şi corectitudinii. Tehnicile de analiză existente sunt fie prea complexe fie prea limitate pentru a fi utile în toate fazele. Din acest motiv, o metodă practică des utilizată în cazul multor sisteme în timp real nu foarte critice este simularea
- ***integrarea analizei de fiabilitate şi toleranță la defecțiuni*** – toleranța la defecțiuni este de o importanță covîrşitoare în sistemele critice. Chiar dacă există la ora actuală cîteva tehnici consacrate în domeniu, multe problemele legate de aplicarea practică a acestora rămîn încă nerezolvate
- ***componente software reutilizabile*** – este general acceptat faptul că reutilizarea codului reduce costurile de dezvoltare precum şi posibilitățile de eroare.

Tehnologia obiectuală pare să aducă în acest sens o contribuție importantă la obținerea de soluții pentru unele probleme dificile din domeniul sistemelor în timp real. Printre motivele care au condus la această concluzie pot fi menționate următoarele:

- obiectele sunt potrivite pentru modelarea sistemelor concurente, autonomia lor logică conduce în mod natural la ideea considerării acestora ca fiind unitatea de bază în cadrul execuțiilor concurente. Prin utilizarea obiectelor paralelismul din lumea reală poate fi exprimat într-un mod mult mai uşor şi natural

- mecanismele de moştenire care sunt prevăzute în majoritatea limbajelor obiectuale încurajează utilizarea componentelor software deja testate, şi care s-au dovedit a fi fiabile şi sigure
- aplicaţiile dezvoltate utilizînd proiectarea şi implementarea orientată pe obiecte sunt mai mai uşor de dezvoltat şi de extins decît cele care utilizează abordarea funcţională, convenţională

Totuşi, pînă în momentul elaborării acestei lucrări, nu s-a realizat încă utilizarea tehnologiei orientate pe obiecte ***pe scară largă*** în proiectarea sistemelor timp real critice şi complexe; este necesară în acest sens îmbunătăţirea metodologiilor existente din punct de vedere al specificării, customizării, catalogării şi regăsirii componentelor software. De asemenea, chiar dacă există numeroase unelte CASE disponibile, (**ObjectGeode** şi **Rational Rose** par să fie unele dintre cele mai puternice în acest sens), acestea sunt deocamdată destul de slab dotate din punct de vedere al semanticii temporale. De exemplu, simulatorul **ObjectGeode** reprezintă o facilitate importantă însă, în cazul aplicaţiilor în timp real distribuite complexe, utilizarea modului de simulare exhaustivă poate deveni practic imposibilă datorită gradului mare de complexitate a acesteia.

Din acest motiv, cel puţin pînă în prezent, majoritatea sistemelor timp real critice au fost şi sunt dezvoltate utilizînd tehnici non-obiectuale, bazate în principal pe descompunerea funcţională. **MGA ACSL/Graphic Modeller** reprezintă în acest sens una dintre cele mai larg răspîndite unelte de modelare şi simulare în domeniul sistemelor de control timp real; din acest motiv, aceasta a fost utilizată şi la dezvoltarea aplicaţiei prezentate în studiul de caz din cadrul capitolului 6.

In mod ideal, pentru dezvoltarea ulterioară a aplicaţiilor de timp real, ***generaţia următoare*** de unelte CASE ar trebui să prezinte următoarele caracteristici:

- ***să suporte principiile de abstractizare şi reutilizare*** – acest lucru a fost deja realizat cu ajutorul tehnologiei obiectuale atît în **Rational Rose** cît şi în **ObjectGeode**; de asemenea, bibliotecile de componente puse la dispoziţie de **MGA ACSL/Graphic Modeller** realizează ceva similar. Este de asemenea necesară introducerea în cadrul metodologiilor a posibilităţii de optimizare a software-ului după integrarea unor componente în scopul atingerii cerinţelor de timp.
- ***să utilizeze şi să combine metodologii deja existente, standardizate, în loc să încerce să inventeze unele noi*** – acest lucru a fost realizat cu UML, MSC şi SDL atît în **Rational Rose** cît şi în **ObjectGeode**

- ***să asiste proiectantul de-a lungul tuturor fazelor de dezvoltare*** - acest lucru a fost realizat în atît să permită analiza cerințelor temporale ale sistemului din punct de vedere al completării, corectitudinii şi al predictabilității – acesta este punctul slab al uneltelor CASE actuale, inclusiv cele prezentate. Parțial acest lucru este rezolvat **în MGA ACSL/Graphic Modeller** şi **ObjectGeode** cu ajutorul simulării; dar chiar dacă simularea reprezintă o unealtă extrem de puternică şi utilă, ea nu poate dovedi predictabilitatea în cazul oricăror sisteme timp real

- ***să asiste programatorul*** la definirea mecanismelor de detectare şi recuperare a erorilor

In concluzie, uneltele existente în prezent aşa numite timp real nu au ajuns pînă în momentul elaborării acestei lucrări la maturitatea care să le permită să fie utilizate pe scară largă în procesul de proiectare a aplicațiilor timp real. In viitor însă, extinderea acestora cu facilitățile prezentate mai sus ar putea conduce însă la existența unor unelte CASE care să fie aplicabile pentru toate categoriile de astfel de aplicații.

4. CONTROLLER-E PROGRAMABILE ÎN APLICAȚIILE DE CONTROL TIMP REAL

4.1. Specificul PLC-urilor

Teoretic, aproape orice tip de calculator poate fi utilizat pentru aplicații de control timp real. Totuşi, avînd în vedere specificul unor astfel de sisteme timp real, care după cum am precizat în capitolele anterioare, se bazează pe calculatoare încorporate, este bine să se utilizeze calculatoare specializate în acest scop. Din această categorie fac parte microcomputerele single-chip-uri, microcontroller-ele, procesoarele digitale specializate (procesoare digitale rapide, calculatoare paralele: transputere, calculatoare RISC pentru utilizarea în aplicații critice), etc [AW88]. Acestea pot fi mult mai uşor adaptate cerințelor aplicației şi utilizarea lor conduce la o reducere considerabilă a efortului de dezvoltare a sistemului de control timp real în cadrul cărora sunt încorporate.

In cazul unui calculator convențional datele sunt în general preluate de la tastatură iar rezultatele sunt vizualizate pe ecran sau la imprimantă. Față de acesta, calculatoarele utilizate pentru controlul proceselor în timp real sunt foarte diferite, în primul rînd datorită faptului că acestea trebuie să interacționeze cu un număr foarte mare şi variat de dispozitive de I/O. Practic, un sistem de control care funcționează în timp real de dimensiuni foarte reduse poate avea un număr de pînă la 20 conexiuni la semnale de I/O; cifre de ordinul a aproximativ 200 de conexiuni sunt absolut normale pentru un sistem de dimensiuni medii.

Chiar dacă este posibilă conectarea acestei cantități de semnale la un calculator convențional, aceasta implică realizarea unor conexiuni non-standard precum şi existența unor dispozitive auxiliare externe. În mod similar, chiar dacă programarea pentru această mare cantitate de semnale de I/O poate fi realizată în limbajele convenționale cum ar fi Pascal, Basic sau C, acestea ar fi utilizate cu un scop pentru care nu au fost inițial proiectate astfel încît rezultatul poate fi destul de incert [Sto92]. De asemenea, operînd în timp real, timpul este o componentă distinctă a strategiei de control: secvențe ca: ***deschide valva ; aşteptă 2,5 s ; porneşte pompa pentru o perioadă de 3 min***, etc. , sunt greu de scris utilizînd limbajele de programare convenționale [SMY00].

Alte cerințe suplimentare, necesare pentru ca un calculator să poată fi utilizat pentru aplicațiile în timp real pot fi sumarizate astfel [SC97]:

- trebuie proiectat astfel încît să poată funcționa în mediu industrial, cu tot ceea ce implică acesta referitor la condițiile de temperatură, umiditate, praf, etc;
- trebuie să fie capabil să utilizeze semnale de I/O digitale la voltajele întîlnite în industrie (de la 24V pînă la 240V), precum şi semnale de I/O analogice; de asemenea, extinderea I/O trebuie să poată fi realizată simplu;
- limbajul de programare trebuie să fie cît mai simplu, ca să poată fi înțeles de persoane cu relativ puțină pregătire în domeniu; programele trebuie să poată fi modificate uşor, în funcție de schimbările care apar în cadrul procesului controlat;
- trebuie avut în vedere că, într-un sistem de control timp real, multe din erorile care apar sunt nu atît erori de programare cît erori rezultate din defecțiunea unor anumiți senzori, swich-uri, etc., astfel încît trebuie să fie posibilă detectarea uşoară a unor astfel de erori de către calculator;
- calculatorul trebuie să fie suficient de rapid pentru a realiza controlul în timp real: acest lucru depinde, după cum a fost precizat anterior atît de viteza calculatorului folosit dar şi de timpul de răspuns caracteristic aplicației;

Avînd în vedere cerințele de mai sus, pentru controlul proceselor industriale complexe, au fost dezvoltate în ultimii ani ***sisteme speciale de calculatoare*** cunoscute sub numele de ***Controller-e Programabile*** **(Programmable Logic Controller - PLC)**, care permit realizarea unor sisteme de control industrial integrate, complexe, şi care să corespundă cerințelor funcționale tot mai crescînde. Acestea sunt proiectate să opereze în medii industriale, cu tot ceea ce implică acestea din punct de vedere al temperaturii, umidității, zgomotelor şi vibrațiilor [Hoh96][Rul97]. Există în prezent o multitudine de firme producătoare de astfel de PLC-uri, dintre care cele mai reprezentative în domeniu sunt:

Allen Bradley	-	PLC-2, PLC-3, PLC-5, SLC-500
Siemens	-	Sinec1, Sinec2
AEG Modicon	-	184, 384, 484, 584, 884, 984
Texas Instruments	-	TI325, TI330, etc.

Termenul de ***sisteme de calculatoare*** utilizat mai sus provine de la ideea că, pe lîngă calculatorul propriu-zis (PLC-ul), producătorii furnizează şi o serie de alte echipamente complementare, în scopul minimizării la maximum a costurilor asociate dispozitivelor şi operațiilor de interfațare.

4.1.1 Caracteristici hardware

Controller-ele programabile constituie o clasă aparte de calculatoare utilizate în mare măsură în cadrul aplicațiilor de control industrial timp real. Din punct de vedere al arhitecturii, PLC-urile sunt o categorie specială de calculatoare utilizată pentru controlul operării proceselor prin intermediul unui program memorat şi utilizînd în acelaşi timp feedback-ul primit de la dispozitivele de I/O [Par95] [Ben94]. Este alcătuit în principal din două părți: ***unitatea centrală*** şi ***interfața pentru I/O***, prin intermediul căreia controller-ul este legat la diversele dispozitive şi senzori exteriori acestuia (Figura 10).

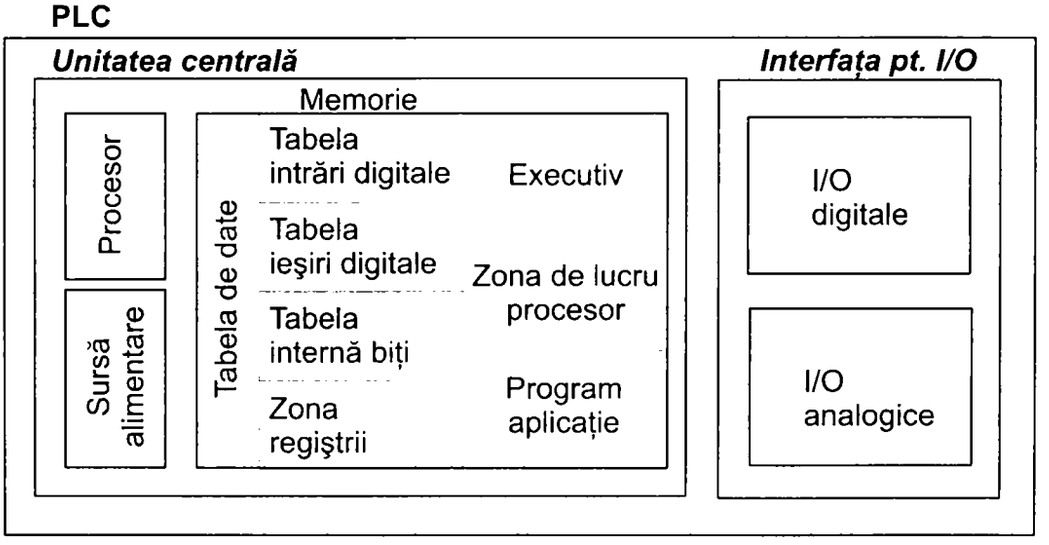


Figura 10: Structura hardware a unui PLC

4.1.1.1. Unitatea centrală

- ***Procesorul*** - realizează toate operațiile matematice şi prelucrările de date prin execuția unei colecții de programe din memorie. In general, această colecție constă dintr-un program *supervizor*, care se află permanent în memorie şi programe *de aplicație*. Programul supervizor (executivul) permite realizarea comunicării cu procesorul via un dispozitiv de programare sau alt periferic, managementul memoriei, monitorizarea I/O, diagnosticarea defecțiunilor hard şi execuția programelor utilizator. Organizarea memoriei precum şi modul în care programele de aplicație sunt executate sub controlul executivului sunt două caracteristici care disting PLC-urile de calculatoarele obişnuite.

- ***Memoria*** - toate PLC-urile au memorie alocată pentru executiv, pentru zona de lucru procesor, pentru tabela de date şi pentru programul de aplicație. Instrucțiunile şi toate datele utilizate de procesor pentru realizarea funcție de control sunt memorate în ***zona de memorie pentru programul de aplicație*** respectiv ***zona de memorie pentru tabela de date***, care împreună formează ***zona de memorie destinată aplicației***. Fiecare controller are o cantitate maximă de memorie destinată aplicațiilor, care este parte a memoriei totale specificată pentru controller. ***Tabela de date*** este în mod funcțional divizată în părți separate pentru ***intrări digitale*** (Input Table), ***ieşiri digitale*** (Output Table), ***tabela internă de biți*** (**flag**-uri) precum ***şi zona de memorie pentru registri*** utilizată pentru memorarea valorilor numerice analogice.

Astfel, ***tabela pentru intrări digitale*** (**Input Table**) constă un şir de biți în care se memorează starea fiecărei intrări digitale care este conectată către sistemul de I/O. ***Tabela pentru ieşiri digitale*** (**Output Table**) la rîndul ei constă dintr-un şir de biți prin intemediul cărora se controlează ieşirile conectate la sistemul de I/O. ***Tabela internă de biți*** (**flag**-uri) reprezintă zona de memorie alocată pentru variabilele binare utilizate în cadrul programului de aplicație. ***Zona de memorie pentru registri*** (**Storage registers Area**) este utilizată pentru memorarea valorilor numerice care pot fi de trei categorii: de intrare (registri de intrare), de ieşire (registri de ieşire) şi respectiv regiştri interni, utilizați pentru memorarea diferitelor valori numerice în cadrul programului de aplicație.

4.1.1.2. Interfața pentru I/O

Interfața pentru I/O reprezintă sistemul prin intermediul căruia diversele dispozitive externe (senzori, actuatori, etc.) sunt conectate la controller. Scopul ei este acela de a realiza condiționarea diferitelor semnale transmise către actuatori respectiv recepționate dinspre senzori. Prin intermediul acestei interfețe, unitatea centrală poate sesiza stări şi/sau măsura diverse mărimi din cadrul procesului controlat: poziții, niveluri, temperaturi, presiuni, tensiuni, curenți, etc. Pe baza acestora pot fi generate comenzi pentru controlul diferitelor dispozitive, ca de exemplu valve, motoare, pompe, etc., sau pentru semnalarea şi/sau tratarea alarmelor.

- ***interfața de I/O digitală (discretă)*** - reprezintă cel mai uzual tip de interfață de I/O. Aceasta se utilizează pentru conectarea dispozitivelor care transmit sau primesc semnale de tip boolean;
- ***interfața de I/O analogică (numerică)*** - spre deosebire de cea discretă, permite transmiterea respectiv recepționarea semnalelor pe mai mulți biți, fie sub formă paralelă (BCD), fie serială, sub forma unui tren de pulsuri. In principiu, grupul de biți transmis sau

recepționat constă din reprezentarea digitală a unei mărimi analogice.

Cantitatea maximă de semnale de I/O, şi implicit a numărului de dispozitive de I/O conectate pentru un anumit tip de controller este dependentă de cantitatea de memorie asociată tabelei de date. Astfel, pentru cazul sistemelor de control timp real de o complexitate mare, care interacționează cu un număr mare de de dispozitive de I/O (de ordinul sutelor), utilizarea unui singur controller nu este în cele mai multe cazuri suficientă, datorită numărului relativ mare de semnale de I/O de prelucrat. In această situație se vor utiliza mai multe controller-e legate în rețea. O astfel de abordare este benefică şi din punctul de vedere al timpului de procesare: se obține astfel reducerea acestuia prin împărțirea încărcării aplicației în mai multe task-uri şi asignarea acestora pe mai multe procesoare.

4.1.1.3. Modul de operare al programului din PLC

Cînd sistemul porneşte, se inițiază execuția programului executiv. In cadrul acestuia procesorul citeşte toate intrările, memorează valorile acestora în tabela de date corespunzătoare, asociată imaginii intrărilor, şi apoi se lansează programul de aplicație. Rezultatele generate în decursul execuției aplicației sunt memorate în tabela de date asociată imaginii ieşirilor. In momentul terminării aplicației, procesorul actualizează toate ieşirile PLC-ului pe baza tabelei de date asociate acestora. Procesul de citire a intrărilor, executării programului şi actualizării ieşirilor este cunoscut sub denumirea de ***proces de scanare***, iar timpul necesar realizării acestuia este cunoscut sub denumirea **de *timp de scanare*** (**scan time**) [War88] [Boa93].

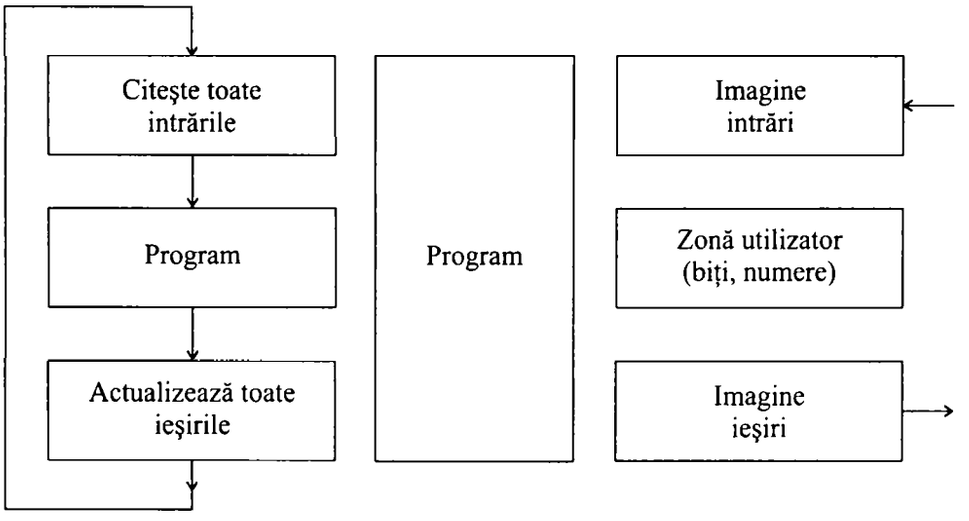


Figura 11: Perioada de scanare şi organizarea memoriei în PLC

Toți producătorii de PLC-uri furnizează o valoare pentru acest timp, de obicei sub forma timpului maxim de execuție ce corespunde fiecărui 1KB de program. In consecință, ca şi modalitate de operare, programul din PLC nu citeşte intrările pe măsură ce acestea apar în program, deoarece aceasta ar fi o risipă de timp, ci deodată, la începutul buclei de scanare. Acestea sunt apoi stocate în memorie, creîndu-se aşa numita imagine a intrărilor (Figura 11). Atunci cînd programul are nevoie de valoarea unei anumite intrări, o citeşte de la locația corespunzătoare imaginii sale din memorie, la valoarea pe care aceasta a avut-o la începutul perioadei de scanare curente [Par95].

În mod analog se tratează şi ieşirile: acestea nu se schimbă instantaneu ci numai după ce zona de memorie corespunzătoare imaginii acestora este schimbată de program. La sfîrşitul perioadei de scanare toate ieşirile sunt actualizate simultan.

Un aspect deosebit de important legat de sistemele de control timp real, caracteristic PLC-urilor, îl reprezintă faptul că perioada de scanare poate avea efect asupra performanțelor programului prin aceea că ea poate cauza întîrzieri nedorite dacă logica programului este inversă scanării: un exemplu de acest fel este ilustrat în Figura 12a).

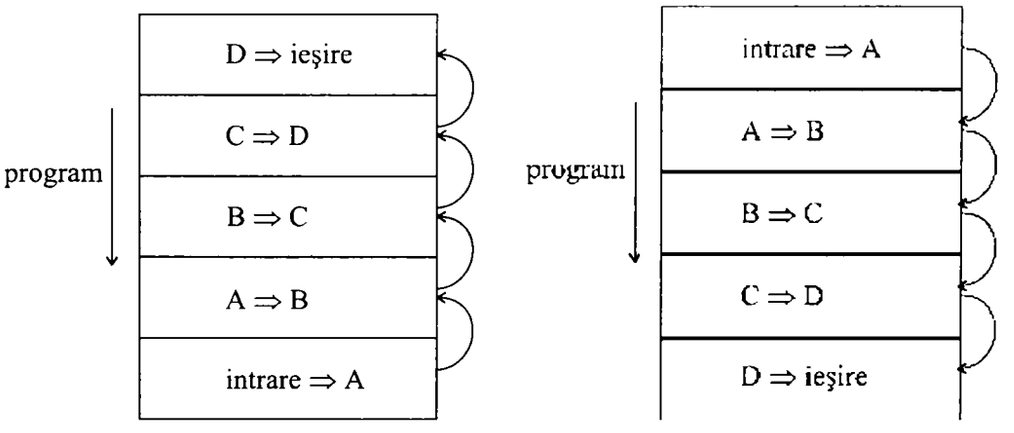


Figura 12: a). Logica programului este inversă scanării

b). Logica programului este în acelaşi sens cu scanarea

Astfel, în cadrul acesteia o intrare (input), cauzează o ieşire (output), trecînd prin fazele intermediare A, B, C, D. În forma de mai sus de realizare a programului, sunt necesare 5 perioade de scanare pentru actualizarea corespunzătoare a ieşirii după primirea semnalului de intrare. Astfel de construcții trebuie evitate în scrierea modulelor, utilizîndu-se secvența corectă, ilustrată în Figura 12b).

In acest caz este necesară doar o singură perioadă de scanare pentru actualizarea ieşirii corespunzătoare. În cazul exemplului de mai sus greşeala poate părea evidentă, dar într-un program complex, de multe ori instrucțiunile care trebuiesc urmate de la primirea unui semnal de intrare şi pînă la actualizarea unei anumite ieşiri nu apar în linii consecutive, astfel încît sunt mult mai greu de depistat. Astfel, în procesul de testare a modulelor trebuie avută în vedere urmărirea şi înlăturarea pe cît posibil a situațiilor de acest gen, prin inspectarea atentă a codului acestora.

Trebuie menționat aici că, în general, producătorii de PLC-uri, furnizează facilități pentru a reduce pe cît posibil influența perioadei de scanare [Lev95]: tipice în acest sens sunt modulele de I/O de mare viteză precum şi posibilitatea de a secționa un program în părți care să utilizeze perioade de scanare diferite. Utilizarea sau nu a acestor facilități sunt aspecte care trebuiesc decise în cadrul procesului de proiectare, funcție de cerințele aplicației de control propriu-zise. In plus, datorită faptului că această metodă bazată pe scanare periodică poate fi necorespunzătoare în aplicațiile care necesită citirea unor intrări foarte rapide, unele PLC-uri furnizează instrucțiuni software care permit întreruperea scanării continue în scopul citirii imediate a unei intrări şi actualizării imediate ieşirilor corespunzătoare.

4.1.2. Arhitecturi pentru sisteme de control automat timp real bazate pe PLC-uri

Sistemele de control timp real pot sau nu să fie şi sisteme distribuite: acest lucru depinde de dimensiunea procesului controlat (cap. 4.1.1) precum şi de extinderea lui geografică. Astfel, structura hardware al unui astfel de sistem de control timp real poate deveni extrem de complexă; un model simplificat utilizat pentru sistemele de control în timp real şi care se potriveşte cu majoritatea situațiilor reale este aşa numitul model pe 5 niveluri [GF86][RD89].

Utilizarea unei astfel de arhitecturi pune probleme deosebite în ceea ce priveşte comunicarea (Figura 13). Analizînd structura modelului din figură, se observă că cerințele sunt diferite pentru diferitele niveluri. Astfel, pentru nivelurile superioare, funcții tipice sunt: transfer de fişiere şi programe, editare de la distanță, achiziție de date de pe nivelurile inferioare, controlul supervizor al nivelurilor inferioare, poştă electronică, etc. Pe aceste niveluri se manevrează cantități mari de date; în cadrul acestora, în mod normal pot fi acceptate întîrzieri.

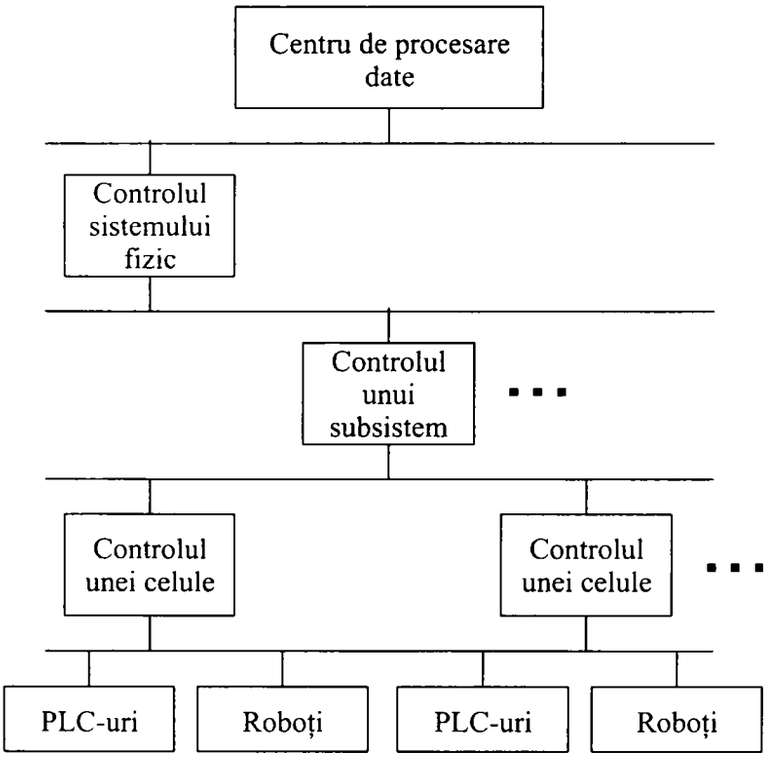


Figura 13: Modelul simplificat al unui sistem de control timp real distribuit

În schimb, pe nivelurile inferioare, transmisia de date trebuie să fie foarte eficientă. Viteza este importantă, dar numai în măsura în care aceasta corespunde cerințelor aplicației: nu sunt necesare viteze de transmisie foarte mari dacă procesul controlat este lent. Cantitatea de informație care se manevrează în acest caz este mică, dar critică din punct de vedere al determinismului [Rod93].

O problemă care rezultă în mod firesc în cazul sistemelor distribuite este aceea a incompatibilității dintre calculatoarele produse de diferiți producători. Pe nivelurile superioare, utilizînd protocoale ***OSI (Ethernet şi TCP-IP)***, această problemă este rezolvată; acestea însă nu pot fi utilizate pentru comunicarea pe nivelurile inferioare, deoarece modelul OSI (pe şapte niveluri) introduce întîrzieri nepermise datorită numărului mare de niveluri. Pentru a îmbunătăți viteza în acest caz, tendințele actuale sunt orientate în două direcții:

- să se dezvolte un protocol redus, care conține numai cîteva din nivelele protocolului OSI inițial (fizic - 1 , legăturii de date - 2 şi aplicație - 7), celelalte niveluri trebuind să fie implementate de utilizator în cadrul aplicației;

- să se renunțe la modelul OSI şi să se dezvolte protocoale speciale pentru comunicarea între dispozitivele de pe nivelele inferioare. Un exemplu în acest caz îl reprezintă aşa numitul **Real Time Manufacturing Message Specification RTMMS** [RIZ90], şi care furnizează mecanisme standardizate pentru transmiterea de mesaje de-a lungul unui proces industrial.

Practic, pentru realizarea unui sistem timp real distribuit, pot fi utilizate trei tipuri conceptuale de rețele [Sta92a] (Figura 14):

- ***rețeaua pentru dispozitive*** (**device network**) : oferă acces rapid la datele sistemului fizic unei game largi de dispozitive; de exemplu, rețeaua **Universal Remote I/O** sau **DeviceNet** [Roc95]

- ***rețeaua de control*** (**control network**) : permite dispozitivelor inteligente de automatizare să folosească în comun informațiile necesare controlului supervizor, interfeței utilizator, configurării dispozitivelor de la distanță, coordonării celulelor de lucru, etc; de exemplu, rețeaua **Data Highway Plus** [Roc95]

- ***rețeaua informațională*** (**information network**) : dă acces sistemelor de calculatoare de pe nivelurile superioare la datele sistemului fizic de bază; de exemplu, rețeaua de tip **Ethernet sau 802.3** [Roc95]

Unele aplicații pot necesita doar un singur tip de rețea, pe cînd altele pot avea nevoie de două sau de chiar toate cele trei tipuri.

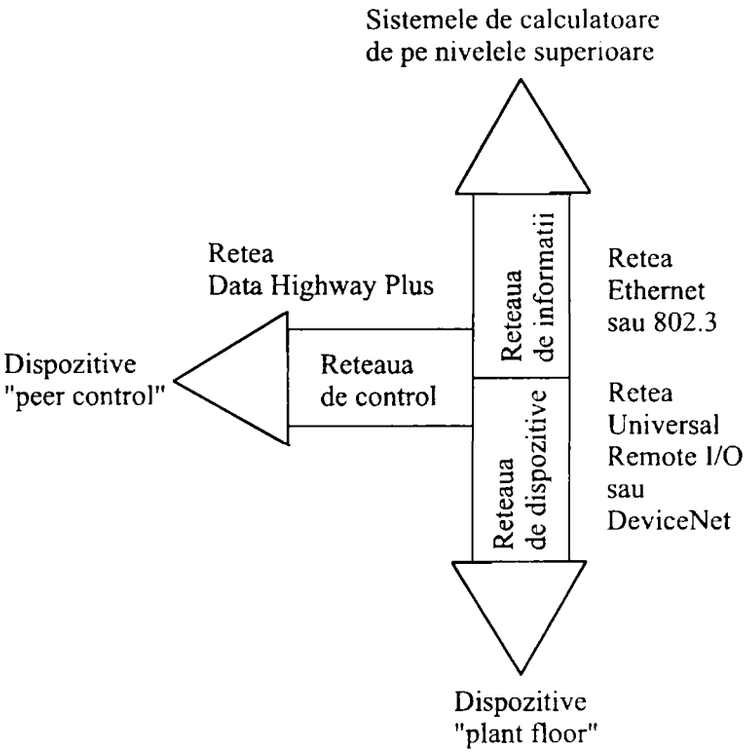


Figura 14: Tipuri conceptuale de rețele

Indiferent de cît de simple sau de complexe sunt cerințele, este de dorit ca dispozitivele din sistem să poată comunica prin intermediul acestor rețele, şi, în plus, să se realizeze şi o comunicare între diferitele tipuri de rețea, cu un minim de interfațare. Din aceste motive, producătorii de controller-e programabile au realizat că funcționalitatea unui sistem de control integrat precum şi costul total sunt, în mare măsură, dictate de uşurința cu care se pot combina componentele acestuia. Din acest motiv, pe lîngă controller-ele propriu-zise, producătorii furnizează şi o gamă completă de componente auxiliare şi de interfațare, de la butoane de comandă şi senzori si pînă la medii software integrate [Con93]. Astfel, se poate asigura o mai bună "curgere" a informației în sistem precum şi un control mai strîns al acestuia, sistemul fiind caracterizat prin ceea ce se numeşte o ***arhitectură deschisă*** (**open architecture**).

4.1.3. Caracteristici software

Dezvoltarea aplicațiilor utilizînd PLC-uri necesită în principal realizarea configurării arhitecturii PLC-ului, adică selecția numărului, tipului şi adreselor utilizate pentru I/O precum şi scrierea şi depanarea programului de aplicație propriu-zis. Programarea PLC-urilor este realizata în general în mod grafic sau

text, utilizînd limbaje specifice, dedicate [Cri90]. De exemplu, pentru cazul aplicațiilor de control timp real a proceselor industriale, gama de facilități necesară fiind relativ mică şi predictibilă, nu este lipsit de sens şi nici prea dificil să l-i se dedice un software de aplicație sau un limbaj special. Majoritatea mediilor de dezvoltarea actuale permit scrierea programelor în mai mult de un singur limbaj [Mic90], cu posibilitatea executării pas cu pas a codului. In plus, unele dintre aceste medii de dezvoltare oferă şi suport pentru realizarea cablajelor pentru I/O şi/sau generarea codului executabil.

Din punct de vedere al software-ului utilizat pentru dezvoltarea aplicațiilor timp real, funcție de capabilitățile şi dimensiunile controller-ului, există la ora actuală trei abordări posibile [Ben94], şi anume:

- software **table-driven**
- software **block-structured**
- limbaje specializate

4.1.3.1. Software table-driven

Structura unui sistem ***table-driven*** este prezentată în Figura 15.

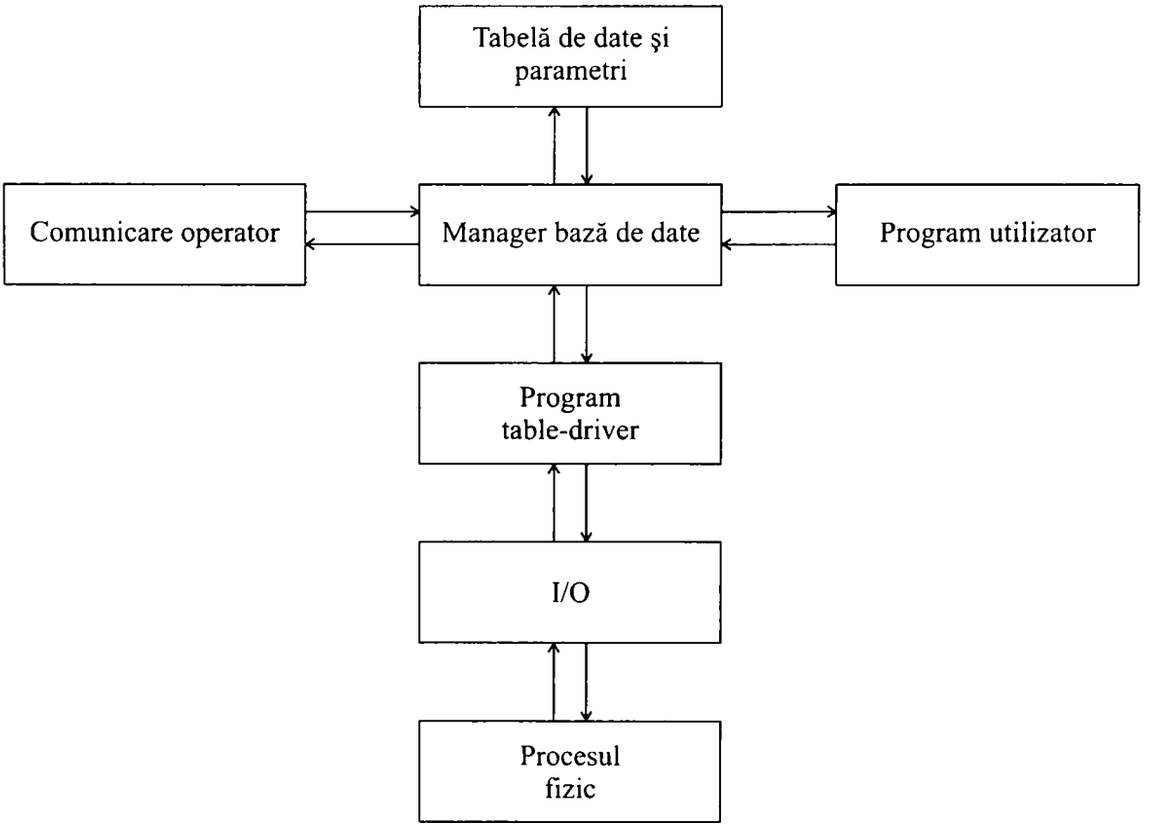


Figura 15: Structura unui sistem table-driven

După cum se observă din Figura 15, un astfel de sistem conține, în principal, un program de control care comunică cu o tabelă de date şi parametri. Programul de control trebuie să conțină toate tipurile de control (bucle, I/O, etc.) care se doreşte să fie folosite; acesta se inserează la configurarea sistemului şi nu mai poate fi schimbat de utilizator. În schimb, utilizatorul poate modifica datele şi eventual parametri prezenți în structurile de control prin intermediul accesului la tabela de date. Anumite sisteme permit utilizatorilor să scrie ulterior programe de aplicație, într-un limbaj uzual (Fortran sau Basic) care să interacționeze cu programul-ul de control.

4.1.3.2. Software block-structured

Dacă software-ul ***table-driven*** este foarte simplu de utilizat, totuşi acesta este foarte restrictiv, în sensul că programul de control este predefinit la configurare. În acest sens, varianta ***block-structured*** aduce un plus de flexibilitate prin utilizarea unor biblioteci de blocuri de funcții [BGM95] (rutine de scanare, rutine pentru controlul PID, rutine pentru ieşiri, funcții aritmetice, blocuri pentru scalare, rutine pentru alarme şi afişare) precum şi a unor modalități de manipulare a acestor blocuri de funcții. Astfel, se poate programa o schemă de control prin interconectarea diferitelor blocuri de funcții şi introducerea parametrilor corespunzători pentru fiecare bloc. Aceasta se realizează în mod curent utilizînd un terminal pe care se pot reprezenta grafic conexinile între blocuri.

Tabelul 4: Blocuri de funcții în controller-e programabile (PLC-uri)

Funcții software:	*De bază*	*Avansate*	*De proces*
	Booleeane Timer-e Counter-e Mutări de date Comparații Aritmetice	Transferuri de blocuri Salturi Fişiere Rotiri de registri Secvențiere Virgulă mobilă	Semnalizare Monitorizare Control PID Comunicare Achiziție Afişare
Funcții hardware:		*PLC-uri mici*	*PLC-uri mari*
	Intrări Ieşiri Timer-e Counter-e Pogram utilizator Timpul / cilclu (per1K)	16 16 8 8 2K 100 ms	4096 4096 256 256 48 K 1 ms

Astfel, pentru aplicațiile de control timp real, care interacționează de regulă cu o mare diversitate de dispozitive de I/O şi conțin numeroase bucle de reglare se preferă utilizarea controller-elor mari, care dispun de un număr relativ mare de I/O precum şi de memorie semnificativ mai mare pentru programele de aplicație (Tabelul 4). In plus, este de dorit ca pentru astfel de sisteme de control să fie disponibile şi funcțiile software de proces (în mod special PID). Criteriul esențial însă care stă la baza alegerii tipului de PLC utilizat în cadrul unei aplicații de control timp real, pe lîngă numărul de I/O şi memorie, îl reprezintă timpul unui ciclu/K de program.

4.1.3.3. Limbajele specializate

Limbajele specializate pentru diferite tipuri de aplicații, variază de la simple interpretoare, care permit interacțiunea cu sisteme de tip "table-driven" sau "functional-block" la limbaje complexe de nivel înalt, care trebuie compilate. In prezent există o largă varietate de limbaje de programare specifice pentru PLC-uri care utilizează fie seturi de instrțiuni simbolice, fie instrucțiuni în limba engleză, în mod analog cu majoritatea limbajelor de programare tradiționale. Cele mai utilizate limbaje specializate pot fi grupate în următoarele categorii [Boa93]:

- limbaje care utilizează diagrame tip ***ladder***
- limbaje care utilizează mnemonice booleene
- limbaje care utilizează blocuri de funcții

Combinația tipică ce apare într-o largă varietate de PLC-uri o reprezintă cea între diagramele ladder şi blocurile funcționale. Bocurile funcționale sunt instrucțiunile care permit utilizatorului să programeze funcții mult mai complexe utilizînd formatul *ladder:* această combinație conduce la existența în principal a 5 categorii de instrucțiuni în cadrul limbajului, şi care includ operații de tipul: ***releu* (relay type), *timer/counter*, *aritmetice*, *manipulări/transfer de date***, precum şi ***operații de control al programului***.

Simbolurile de bază utilizate în cadrul acestor categorii de limbaje sunt ***contactele*** (**contacts**) şi ***conductoarele*** (**coils**), iar programarea se realizează la nivel de ***rung***, asimilat în acest caz cu o linie de program (Figura 16).

Astfel, un rung constă dintr-un set de condiții de intrare, reprezentate printr-o combinație de contacte, precum şi o instrucțiune de ieşire, la sfirşitul rung-ului, reprezentată printr-un conductor, care va fi sau nu ***energizată*** funcție de condițiile din intrare. Fiecare contact sau conductor este referit printr-o adresă

care identifică intrarea evaluată sau ieşirea controlată, aceste adrese referind practic o locație din tabela de intrare respectiv din tabela de ieşire a PLC-ului.

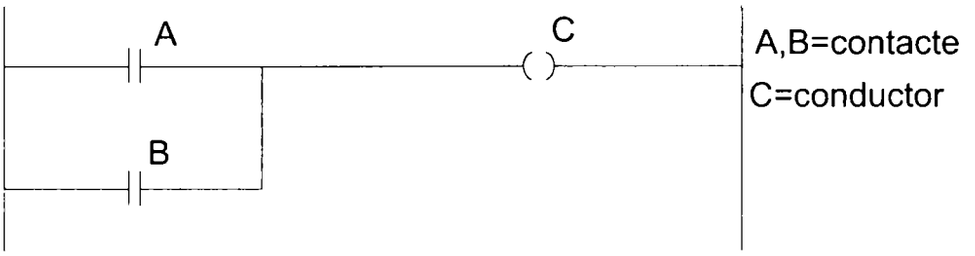


Figura 16: Structura unui *rung*: if A or B then C

Programele dezvoltate utilizînd limbajele specializate sunt în general construite pe o structură modulară, fiind împărțite în mai multe blocuri funcționale care sunt văzute ca şi subrutine apelate din programul principal. Avantajul subrutinelor rezultă bineînțeles într-o minimizare a efortului de programare şi realizarea unor programe mai uşor de urmărit. De asemenea, pot fi concentrate în subrutine separate prelucrări pentru subsisteme asemănătoare (sau chiar identice), în măsura posibilităților, sub o formă paramerizată.

Un exemplu de astfel de limbaj specializat, utilizat pe larg în cadrul controller-elor de tip Allen Bradley [Roc94a], este **APS (Advanced Programming Software)** [Roc94b] [Roc94c]. Acesta îmbină caracteristicile ladder cu programarea funcțională şi este deosebit de potrivit în utilizarea pentru sistemele de control timp real, el incluzînd, pe lîngă funcțiile de bază, şi toate funcțiile software de proces prezentate în Tabelul 4.

De exemplu, una dintre cele mai importante funcții de proces pusă la dispoziție de **APS** o reprezintă funcția PID prin intermediul cărora se poate realiza controlul PID (proporțional-integral-derivativ) al marimilor controlate prin intermediul diferitelor regulatoare [PH96], printr-o simplă modificare a parametrilor unui regulator general.

Pe lîngă facilitățile de limbaj propriu-zis, **APS** furnizează şi de un mediu de dezvoltare al programelor deosebit de prietenos; cîteva dintre caracteristicile puse la dispoziție de către acesta sunt următoarele:

- permite crearea, editarea şi monitorizarea deosebit de facilă a programelor atît on-line şi off-line
- permite forțarea biților de intrare/ieşire în scopul testării anumitor scenarii

- permite realizarea diferitelor tipuri de căutări
- afişează documentaţia programului atît on-line cît şi off-line
- permite stocarea programelor pe disc în scopul arhivării sau al transferului (downloading) către alte procesoare
- permite tipărirea la imprimantă a textului sursă al programului

APS permite programarea procesoarelor Allen Bradley din gama **SLC 500** utilizînd un terminal grafic sau orice calculator personal compatibil IBM PC. In cazul utilizării mediului Microsoft Windows, **APS** operează în cadrul unei ferestre DOS, astfel încît poate rula în cadrul aceleeaşi sesiuni de lucru cu alte programe.

Multe PLC-uri, printre care şi Allen Bradley SLC-5/03, permit de în plus împărţirea programului în blocuri executabile şi prin care se permite saltul peste anumite linii (rung-uri) de program, dacă instrucţiunile de la începutul blocului sunt adevărate (instrucţiunea MCR - Master Control Relay); această facilitate poate îmbunătăţi timpul de scanare pentru aplicaţiile care necesită o viteză sporită. Totuşi, utilizarea unor astfel de instrucţiuni este periculoasă, în sensul că ea poate crea confuzii în procesul de întreţinere a programului, dacă acesta nu este documentat corespunzător.

In plus, ca şi o observaţie, trebuie menţionat că ***anumite procesoare***, ca de exemplu SLC-5/02 dispun de instrucţiuni suplimentare asociate cu funcţia STI (Selectable Timed Interrupt), pentru activarea (STE Selectable Timed Enable), dezactivarea (STD Selectable Timed Disable) sau iniţierea (STS Selectable Timed Start) acesteia. Există, de asemenea, şi instrucţiunea INT, asociată cu întreruperile de I/O care poate fi utilizată în cazul în care se doreşte o viteză de reacţie sporită, şi care nu poate fi atinsă utilizînd ciclul obişnuit de scanare.

In concluzie, se poate spune că dezvoltarea unui program de aplicaţie pentru un sistem timp real utilizînd PLC-uri implică mai mult decît simpla scrierea acestuia; asignarea adreselor pentru I/O şi elementele interne, testarea funcţionării corecte a acestuia precum şi optimizarea acestuia sunt paşi care trebuiesc parcurşi înaintea adoptării unei soluţii definitive [Shi96].

4.2. Paradigma Cy-Clone

4.2.1 Concepte fundamentale privind abordarea ciclică

In ultimii ani a existat o reținere în utilizarea abordării ciclice în sistemele timp real. Acest lucru se datorează în principal faptului că simplificarea proprietăților fundamentale nu prezenta o provocare semnificativă pentru domeniul academic. Aplicațiile practice au dovedit însă că această viziune este total greşită pentru că, pînă acum utilizarea abordării ciclice poate reduce semnificativ complexitatea aplicațiilor.

Un model pe care îl propun în acest sens [Law92] este aşa numitul **Cy-Clone (Clone of the Cyclic Paradigm)**, şi el se bazează pe principiul ***resource adequacy*** prezentat în cadrul paragraful 2.1.4. ***Ideea de bază a paradigmei Cy-Clone este aceea că există suficiente resurse în sistem astfel încît să se garanteze că toate operațiile vor fi executate în timp util.***

Bazat pe această paradigmă, logica aplicației de programat trebuie împărțită în patru părți, care se vor repeta periodic, cu perioada ***dT***, după cum este ilustrat în Figura 17.

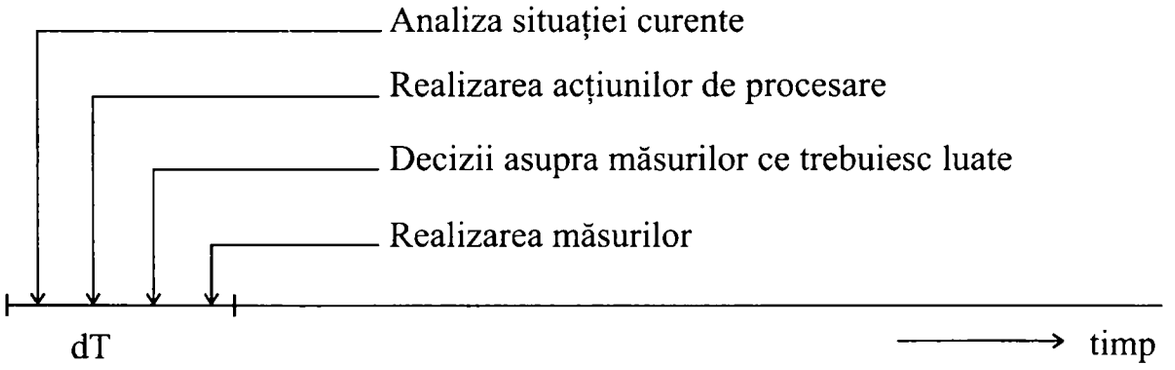


Figura 17: Ciclul de bază conform paradigmei **Cy-Clone**

Proprietatea fundamentală de timp în acest caz o reprezintă intervalul ***dT***, care stabileşte frecvența execuțiilor ciclice. Intervalul ***dT*** trebuie să fie suficient de lung pentru a permite terminarea tuturor operațiilor aferente unei aplicații în cadrul unei perioade, dar şi suficient de scurt pentru a asigura stabilitatea şi cerințele de timp ale sistemului; în consecință, ***dT*** trebuie ales pe baza cunoaşterii dinamicii procesului controlat [OR94].

Procesarea periodică poate fi în acest caz condiționată sau necondiționată. Pe baza paradigmei **Cy-Clone** se pot concepe diferite modele de implementare, atît pentru sisteme centralizate cît şi pentru sisteme distribuite. Implementările pot furniza, în cazul în care este util, modalități de modificare controlată a intervalului ***dT*** în decursul execuției. In mod normal, în practică se poate utiliza un controller pentru implementarea propriu-zisa.

Astfel, se observă că în cadrul paradigmei ciclice se utilizează controlul temporal: la puncte fixe de timp se determină dacă anumite operații trebuiesc activate sau nu, în funcție de îndeplinirea anumitor condiții la momentul respectiv (sistem **TT**). In mod normal, se utilizează un task trigger care captează în mod regulat starea curentă. Acest task trigger este un task periodic care evaluează condiția de trigger asupra unui set de parametri şi transmite un semnal de control care activează un task de aplicație. Deoarece starea curentă este captată la frecvența task-ului trigger, numai acele stări cu o durată mai mare ca perioada de achiziție a task-ului trigger pot fi observate.

In plus, task-ul trigger periodic generează o întîrziere administrativă (administrative overhead) în sistemele timp real de time-triggered **TT**. Bazat pe aceste considerente, perioada task-ului trigger trebuie să fie semnificativ mai mică decît ***laxity-ul*** (diferența între timpul de deadline şi timpul de execuție) pentru orice tranzacție în timp real care poate fi activată de un eveniment extern. Dacă ***laxity-ul*** tranzacțiilor este foarte mic (de exemplu, <1ms), întîrzierea asociată utilizării unui task trigger devine intolerabilă [Kop97]; astfel de sisteme timp real nu pot fi modelate utilizînd abordarea ciclică.

4.2.2. Abordarea ciclică în cadrul sistemelor distribuite

Conceptele prezentate mai sus în cadrul paradigmei **Cy-Clone** pot fi extinse şi în cazul sistemelor timp-real distribuite, prin introducerea aşa numitului **Cy-Clone distribuit**. Intr-o astfel de abordare, fiecare nod din cadrul sistemului distribuit constă dintr-un controller distinct, care operează conform ciclului de bază prezentat în Figura 17. Nodurile operează în paralel, frecvența execuțiilor ciclice putînd diferi de la un nod la altul, fiecare avînd însă proprietăți temporale bine definite.

Nodurile din cadrul sistemului distribuit pot sau nu să fie conectate către senzori şi actuatori, fiecare nod avînd însă un rol distinct în cadrul sistemului. Pentru sincronizare, este necesară folosirea unui ceas global de timp real, fie extern (de exemplu, via GPS), fie intern, bazat pe frecvența ceasului mediului de comunicare utilizat.

In ceea ce priveste comunicarea, fiecare nod conține o tabelă de comunicare cu alte noduri (TC), şi care este parte a tabelei de date a controller-ului respectiv. Aceasta conține de obicei o cantitate de informație relativ redusă în

comparație cu cantitatea de informație care este tratată la periferia sistemului. Suma tuturor tabelelor asociate tuturor nodurilor formează aşa numita ***Tabelă de date timp real***; practic, fiecare nod din cadrul sistemului distribuit are o fereastră în această tabelă.

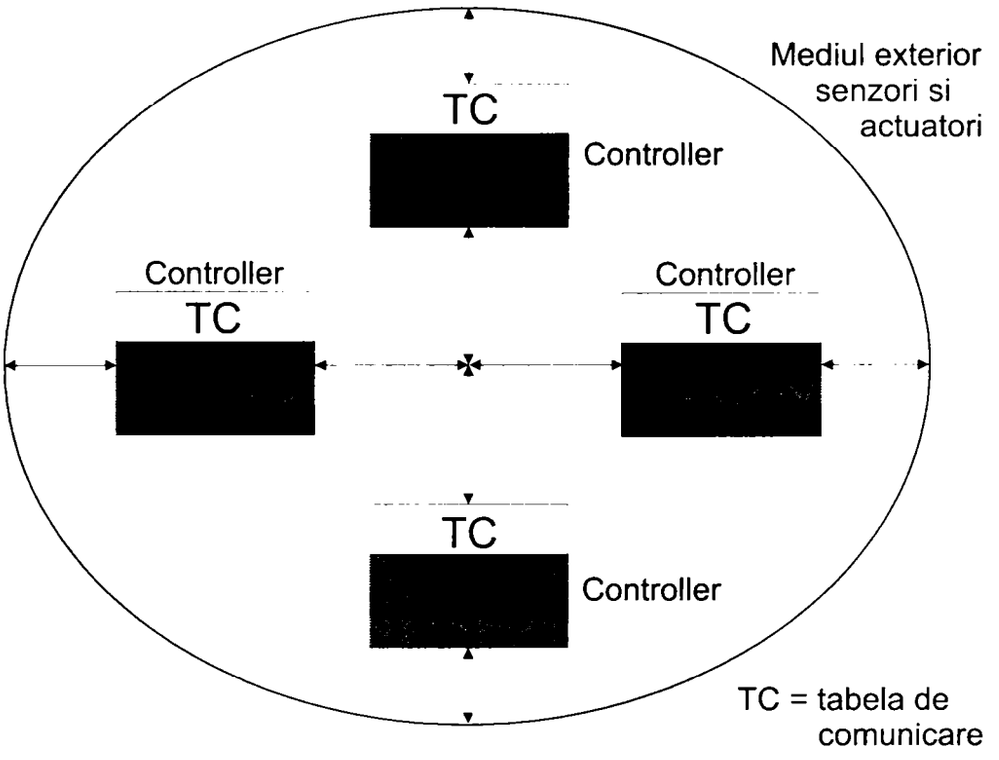


Figura 18: Cy-Clone distribuit – modelul conceptual

Comunicarea datelor între noduri se realizează practic prin utilizarea ***Tabelei de date timp real*** şi care reprezintă zona de date comune a sistemului distribuit, fiind accesibilă tuturor nodurilor. Modul de referire respectiv notația folosită pentru accesul datelor din cadrul acesteia este în general similară, pentru majoritatea PLC-urilor, cu maniera de referire a datelor din cadrul tabelei de date proprii. Astfel, pentru un anumit nod, celelalte noduri furnizează date în mod periodic, în aceeaşi manieră în care sunt furnizate periodic datele de la senzorii externi. Astfel, este util ca, din punct de vedere conceptual dar şi tehnic (software), accesul la date (comune sau nu) să se realizeze într-o manieră unitară.

Varianta **Cy-Clone distribuit** furnizează concepte pentru tratarea aspectelor speciale legate de complexitatea sistemelor de control timp real distribuite fiind în acelaşi timp capabilă să exploateze proprietățile de predictibilitate şi determinism utilizate în cadrul paradigmei Cy-Clone originale.

4.2.3. Avantajele şi dezavantajele paradigmei Cy-Clone

Avantajele paradigmei Cy-Clone

- ***existența unei discipline temporale stricte*** – un avantaj deosebit de important care rezultă în urma aplicării paradigmei ciclice este faptul că există o disciplină temporală foarte bine definită şi care permite realizarea unui control hardware sincron. In acest sens, informația este comunicată către şi dinspre ciclul de bază la momente fixe de timp. Sincronizarea poate fi realizată utilizînd timpul relativ, local, sau, mult mai precis, utilizînd sistemul GPS, rezultînd soluții deterministe şi foarte exacte din punct de vedere al timpului.In mod evident, nu toate aplicațiile critice de timp real necesită o astfel de sincronizare hardware. In timp ce intervalul de bază de timp ***dT*** furnizează o evoluție acceptabilă a procesării şi asigură stabilitatea sistemului, relaxarea sincronizării se poate dovedi extrem de utilă în reducerea complexității planificării pentru sistemele distribuite. In acest sens, rezultatele sunt utilizate pe baza celei mai bune informații disponibile.

- ***mai bună detectare şi localizare a erorilor*** – după cum a fost ilustrat în cadrul capitolului 2, marele avantaj al abordării ***event-triggered (ET)*** în ceea ce priveşte tratamentul erorilor este flexibilitatea şi răspunsul imediat; pentru cazul ***time-triggered (TT)*** însă se poate realiza o mai bună detectare şi localizare a erorilor [Kop98]. Un alt avantaj ce rezultă în urma aplicării soluției bazate pe controlul temporal o reprezintă posibilitatea de a trata defecțiunile hardware şi software apărute într-o manieră uniformă. Astfel, în cazul utilizării abordării ciclice, o defecțiune poate avea doar efecte locale în timp, dar sistemul se va stabiliza după cîteva perioade de timp succesive. Aceasta reprezintă o proprietate deosebit de importantă, şi este extrem de greu de obținut în cazul în care se utilizează o manieră probabilistică de execuție a task-urilor, bazată pe evenimente (***ET***).

- ***obținerea predictabilității şi determinismului pentru sistemele timp real*** – datorită controlului temporal strict, soluția bazată pe paradigma **Cy-Clone** reprezintă o abordare pertinentă şi naturală pentru a obține predictibilitate şi determinism în sistemele de timp real critice, cu atît mai mult cu cît, datorită creşterii puterii de procesare a calculatoarelor actuale, problemele legate de asigurarea resurselor necesare nu mai sunt atît de critice.

Dezavantajele paradigmei Cy-Clone

- ***lipsa de flexibilitate în comparație cu sistemele event-triggered ET*** – în cazul utilizării paradigmei **Cy-Clone**, ordinea desfaşurării tuturor acțiunilor precum şi momentele de timp asociate acestora sunt definite şi cunoscute în prealabil

- ***nu poate fi aplicată oricărui tip de sistem*** – în cazul în care constrîngerile de timp impuse sunt mai mici decît ciclul de bază ***dT***, trebuie luată în considerare abordarea bazată pe evenimente ***ET***

4.2.4. Utilizarea paradigmei Cy-Clone si a PLC-rilor in dezvoltarea aplicațiilor de control timp real

Avînd în vedere modul de operare caracteristic PLC-urilor, pe baza perioadei de scanare, pentru proiectarea sistemelor timp real cu ajutorul controller-elor programabile se poate utiliza paradigma ciclică prezentată la capitolul anterior. Acest lucru se bazează pe faptul că, în principiu un sistem de monitorizare şi control timp real realizează în mod tipic un set de acțiuni care se repetă periodic:

- monitorizarea intrărilor
- controlul ieşirilor
- acțiuni de procesare (ex: controlul buclelor de reglare)
- tratare alarme
- înregistrare date (ex: pentru trasarea ulterioară a diagramelor de evoluție în timp), etc.,

Ca atare, un astfel de sistem poate fi văzut ca fiind alcătuit dintr-o buclă care se repetă în mod ciclic, la intervale regulate de timp.

In contextul PLC-urilor, abordarea ciclică presupune existența unui program principal în cadrul căruia modulele sunt implementate ca şi subrutine acestuia. Acest tip de structură este foarte uşor de programat, dar impune constrîngeri severe asupra tuturor modulelor: toate modulele trebuie să se execute într-o perioadă de timp mai scurtă decît intervalul de timp ***dT***, care stabileşte frecvența execuțiilor ciclice şi este impus de aplicația propriu-zisă (Figura 17).

In consecință, structura programului trebuie să se bazeze pe partiționarea aplicației în module pe baza funcționalității şi a constrîngerilor de timp, în cazul

în care este necesar, şi care sunt cuplate împreună utilizînd o structură de control, ca cea prezentată în Figura 19.

```
DO forever
DO during dT
CASE
 mode1:
  WITH sensors AND actuators DO
   [EVERY Nth dT ]
   BEGIN
      operation1();operation2();
      ....
      IF situation_predicate THEN
       ....
        operationi();operationj();
       ....
      IF situation_predicate THEN
       ....
       operationk();operationm();
       ...
      .....
   END
 mode2:
  WITH sensors AND actuators DO
   [EVERY Nth dT ]
   BEGIN
      operationp();operationq();
      ....
      IF situation_predicate THEN
      ....
      operationr();operationv();....
     .....
   END
........
 startup_mode:
  WITH sensors AND actuators DO
   [EVERY Nth dT ]
   BEGIN
     .....
   END
ENDCASE
```

Figura 19: Structura de bază a programului în PLC

Este evident că structura de bază a software-ului prezentat mai sus corespunde cu conceptul utilizat în cadrul abordării ciclice, pezentat în Figura 17. In alegerea structurii hardware şi implicit al PLC-ului corespunzător rolul hotărîtor îl are intervalul ***dT***, care determină frecvența execuțiilor ciclice.

Astfel, pe baza cerințelor temporale impuse de aplicație se poate calcula valoarea maximă pe care o poate avea intervalul ***dT*** pentru care sunt satisfăcute toate constrîngerile de timp ale aplicației.

In procesul de determinare a intervalului ***dT***, se porneşte de la cazul prezentat în Figura 20. In cadrul acesteia, modificarea unei intrări, notate cu A, şi care ar trebui să conducă la modificarea unei anumite ieşiri, notate cu B, apare imediat după începutul ciclului de scanare curent; acest lucru implică faptul că sesizarea acestei modificări va fi realizată numai la începutul buclei de scanare următoare, iar ieşirea B va fi modificată abia la sfirşitul acesteia. Deci, conform acestui exemplu, putem spune că în cazul cel mai defavorabil sunt necesare ***2 intervale dT*** pentru actualizarea corespunzătoare a iesirii B, deci acțiunea ***A implică B*** va dura, în cazul cel mai defavorabil ***2*dT*** . Evident, acest lucru este valabil numai dacă logica aplicației de programat nu este inversă scanării (Figura 12), altfel numărul de intervale ***dT*** poate fi mai mare.

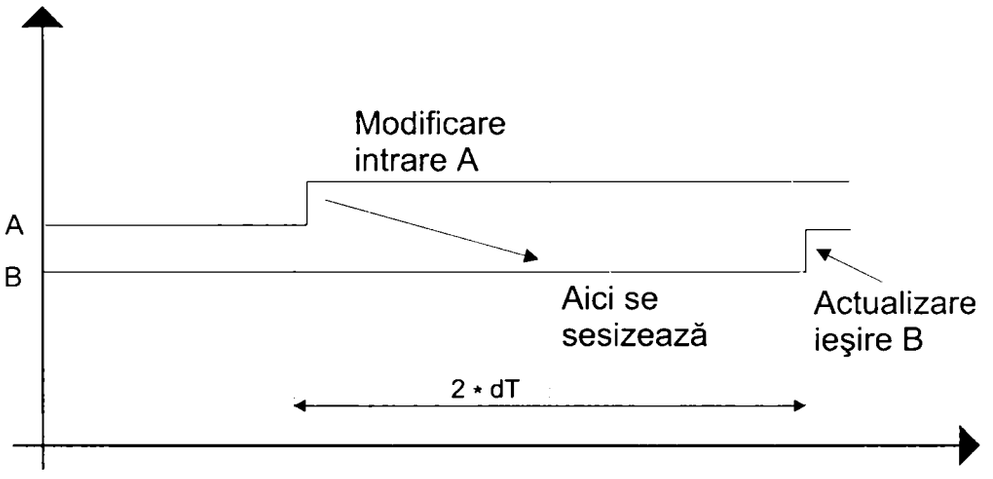


Figura 20: Durata unei acțiuni ***A implică B*** în abordarea ciclică

In consecință, în alegerea arhitecturii hardware utilizate în cadrul unui sistem de control timp real care utilizează abordare ciclică şi PLC-uri trebuie să se respecte următoarele etape:

- determinarea, pe baza analizei cerințelor aplicației date, a acțiunii cu constrîngerea de timp cea mai mică; se notează cu t_lim_{min} intervalul de timp în cadrul căruia aceasta trebuie să se execute

- calcularea intervalului dT_{max} necesar pentru ca acțiunea corespunzătoare să se încadreze în intervalul t_lim_{min}, după formula rezultată în urma analizei realizate pentru Figura 20:

$$dT_{max} \leq t\_lim_{min} / 2 \qquad \text{(r4.1)}$$

- evaluarea intervalului $\boldsymbol{dT_{max\_PLC}}$ pentru un anumit PLC funcție de caracteristicile acestuia. Valoarea $\boldsymbol{dT_{max\_PLC}}$ depinde de mărimea programului din PLC precum şi de viteza procesorului; astfel, se poate face o evaluare grosieră pentru cazul cel mai defavorabil bazată pe valorile date de producător pentru timpul de scanare/K program $\boldsymbol{t_{scan/K}}$, şi dimensiunea maximă a programului de aplicație $\boldsymbol{dim_{max\_progr}}$, rezultînd:

$$dT_{max\_PLC} = dim_{max\_progr} * t_{scan/K} \qquad \text{(r4.2)}$$

- compararea valorii $\boldsymbol{dT_{max\_PLC}}$ cu $\boldsymbol{dT_{max}}$ impus de specificațiile aplicației date; astfel, pentru garantarea îndeplinirii constrîngerilor de timp minime trebuie să fie îndeplinită relația:

$$dT_{max\_PLC} \leq dT_{max} \qquad \text{(r4.3)}$$

- determinarea valorii pentru constrîngerile de timp minime care pot fi satisfăcute în cazul cel mai defavorabil pentru PLC-ul cu caracteristicile date. Astfel, în urma înlocuirii valorilor pentru $\boldsymbol{dT_{max}}$ şi $\boldsymbol{dT_{max\_PLC}}$ în cadrul relației (r4.3) rezultă:

$$dim_{max\_progr} * t_{scan/K} \leq t\_lim_{min} / 2 \qquad \text{(r4.4)}$$

  In consecință, în cazul cel mai defavorabil, pentru un PLC cu anumite caracteristici date, ***pot fi garantate*** constrîngerile de timp cu o durată mai mare decît:

$$t\_lim_{min} \geq 2 * dim_{max\_progr} * t_{scan/K} \qquad \text{(r4.5)}$$

Valorile $\boldsymbol{dim_{max\_progr}}$ şi $\boldsymbol{t_{scan/K}}$ sunt, după cum s-a precizat anterior, furnizate de producători şi deci cunoscute în prealabil. Calculele de mai sus reprezintă o evaluare grosieră a timpului limită minim $\boldsymbol{t\_lim_{min}}$, în anumite condiții acesta mai putînd fi însă redus.

Abordarea ciclică poate fi aplicată atît în mod centralizat (utilizînd un singur PLC) cît şi în mod distribuit (mai multe PLC-uri). In primul caz, este necesar ca toate procesările impuse de aplicație să poată fi realizate în limitele intervalului $\boldsymbol{dT_{max}}$, conform celor prezentate anterior. In cazul în care resursele nu sunt suficiente pentru realizarea acestui deziderat, adică relația (r4.3) (şi implicit (r4.4)) nu poate fi îndeplinită pentru un anumit caz concret, există două modalități de a asigura resursele necesare (***resource adequacy***):

- alegerea unui PLC mai performant, cu un timp de scanare/K mai scăzut, astfel încît, pentru toate constrîngerile temporale din sistem să fie satifăcută relația de mai sus

- utilizarea modelului **Cy-Clone** distribuit: logica aplicației de programat se va împărți pe mai multe PLC-uri (noduri în sistemul distribuit) astfel încît în cadrul fiecărui nod să se garanteze îndeplinirea cerințelor temporale impuse. Uneori aceste noduri pot opera la frecvențe diferite, însă ele trebuie să se sincronizeze funcție de un un timp de bază pentru a asigura funcționarea corectă a aplicației; în acest caz, trebuie considerat în plus şi timpul de comunicare între nodurile 1 şi 2, t_{com12}, astfel:

$$t\_lim_{min} \geq 2 * dim_{max\_progr\_nod1} * t_{scan/K\_nod1} + t_{com12} + 2 * dim_{max\_progr\_nod2} * t_{Bscan/K\_nod2} \quad \text{(r4.6)}$$

După cum se poate observa, rezultă în acest caz un timp mai mare decît cel din relația (r4.5), pentru cazul în care acțiunile A şi B (Figura 20) sunt localizate pe procesoare diferite; din acest motiv, acțiunile cu constrîngeri de timp stricte, de tipul ***A implică B***, trebuie grupate pe cît posibil pe acelaşi procesor. Timpul de comunicare t_{com12} poate fi, în anumite cazuri neglijabil fată de restul termenilor din relația (r4.6), în special dacă sunt utilizate rețele de comunicare speciale timp real, de tipul celei de dispozitive (Figura 14).

4.3. Concluzii

Avînd în vedere cerințele tot mai crescînde ale sistemelor timp real în raport cu predictabilitatea, în practică este esențială găsirea unor soluții cît mai simple şi care să furnizeze ***garanția*** că toate procesările critice vor fi realizate în timp util. Funcție de cerințele şi de natura aplicației propriu-zise, diferite soluții pot fi mai mult sau mai puțin viabile la un moment dat. Dintre acestea, soluțiile bazate pe evenimente (***ET***), chiar dacă mai flexibile, prin natura lor ne-deterministă impun însă anumite limitări în realizarea unor proprietăți cruciale cum ar fi predictabilitatea, fiabilitatea, sau testabilitatea sistemelor timp real. In plus, pe măsură ce complexitatea sistemelor creşte, o atenție sporită trebuie acordată problemelor de integrare.

In acest capitol a fost abordată problema posibilității utilizării PLC-urilor şi a abordării ciclice în sistemele de control timp real. Datorită modului de operare caracteristic PLC-urilor, s-a demonstrat că acestea se pretează foarte bine la utilizarea în cadrul sistemelor de control **time-triggered *TT***, şi care sunt

proiectate utilizînd paradigma **Cy-Clone**. In plus, integrarea sistemelor ***TT***, cum este cazul sistemelor în care sunt utilizate PLC-uri şi abordarea ciclică, este mult mai uşor de realizat decît integrarea sistemelor ***ET***.

S-a demonstrat de asemenea faptul că, utilizînd o astfel de abordare, este posibilă atingerea dezideratelor de predictabilitate şi determinism în sistemele timp real: aceasta însemnînd în esență că, dîndu-se un set de cerințe, se poate garanta că sistemul le îndeplineşte întotdeauna.

Există o mulțime de critici care au fost aduse aplicațiilor care adoptă abordarea ciclică: dintre dezavantajele cunoscute ale acestora sunt inflexibilitatea, şi faptul că nu orice tip de sistem timp real poate fi proiectat şi modelat utilizînd acest mod de abordare. Totuşi, în prezent există o largă categorie de aplicații pentru care soluția propusă este perfect viabilă şi pentru care aceasta furnizează o bază cunoscută pentru obținerea predictabilității şi determinismului: condiții esențiale pentru sistemele în timp real. Mai mult, dacă sunt proiectate corespunzător, aceste aplicații pot furniza o comportare în timp care poate fi reprodusă (determinism reproductibil).

Dacă însă, datorită constringerilor de timp foarte mici, aplicația nu poate fi proiectată utilizînd controlul temporal ***TT*** şi deci abordarea ciclică, utilizarea întreruperilor trebuie avută în vedere din considerente de performanță. Dar, după cum a fost deja precizat, prin utilizarea **event-triggered *ET*** predictablitatea şi determinismul sunt deziderate greu de atins. Este foarte clar că o astfel de soluție care permite schimbarea în mod dinamic a ordinii proceselor dintr-un sistem va rezulta într-un sistem ne-deterministic. Astfel, rezultatele probabilistice obținute de Liu şi Layland privind algoritmul de planificare rate-monotonic pentru un număr arbitrar de procese periodice pot fi utilizate doar pentru dovedirea predictabilității, nu şi al determinismului [LL73].

5. METODOLOGIE DE PROIECTARE A APLICAȚIILOR TIMP REAL UTILIZÎND PLC–URI

5.1. Concepte generale

Sistemele de control timp real reprezintă o categorie aparte de sisteme care îmbină caracteristicile sistemelor software în general cu acelea ale sistemelor de control automat în particular. Atît în domeniul ingineriei programării **IP** [Kav92] cît şi în domeniul ingineriei controlului automat **ICA** există o serie de metodologii consacrate[KY95], însă fiecare dintre acestea adresează o categorie specifică de aplicații:

- metodologiile sau modelele din domeniul **IP** se referă la dezvoltarea aplicațiilor software în general, fără a conține de cele mai multe ori aspecte specifice controlului timp real; în plus, unele dintre acestea sunt mult prea complexe şi dificil de utilizat în aplicațiile practice, reale. O selecție a celor mai cunoscute metodologii din cadrul acestui domeniu a fost prezentată în paragraful 1.3.2;

- metodologiile sau modelele din domeniu **ICA** se referă în general la aspecte legate de analiza şi proiectarea algoritmilor de control propriu-zişi şi nu la aspecte legate de dezvoltarea sistemului în ansamblul său [FB95].

Complexitatea crescîndă a sistemelor de control timp real conduce la necesitatea utilizării combinate a tehnicilor din domeniul ingineriei programării **IP** precum şi din domeniul ingineriei controlului automat **ICA** în cadrul procesului de proiectare şi implementare a acestora. In consecință, avînd în vedere domeniul multidisiplinar al sistemelor de control timp real, este necesară dezvoltarea de noi metodologii, simple şi uşor de aplicat în practică, specifice acestui domeniu; în plus, pentru a lua ce este mai bun din ceea ce există în acest domeniu, acestea ar fi ideal să se bazeze pe combinarea metodologiilor deja existente în cadrul domeniilor **IP** şi **ICA**.

In continuare se va prezenta o metodologie de proiectare a aplicațiilor de control timp real bazată pe paradigma ciclică prezentată în cadrul paragraful 4.2. Metodologia încearcă să furnizeze un set de concepte şi criterii care pot fi utilizate în procesul de dezvoltare în ansamblu a unei aplicații de control timp real. Metoda este orientată către aplicații care au o puternică componentă de timp real: control, achiziție şi prelucrare masivă de date şi se bazează pe utilizarea PLC-urilor în cadrul procesului de implementare propriu-zisă a programului de control.

Utilizarea PLC-urilor şi al controlului temporal ***TT***, specific paradigmei ciclice, asigură comportarea predictibilă a sistemului în orice situație, şi implicit reprezintă o premiză pentru obținerea şi a altor caracteristici printre care cele mai importante sunt fiabilitatea, testabilitatea, verificabilitatea precum şi posibilitatea de întreținere cît mai uşoară a sistemului.

In plus, metodologia încearcă să furnizeze, pe lîngă criteriile de structurare, tehnicile de dezvoltare şi modalitățile de testare prezentate, şi o combinație de unelte software (CASE tools) cu ajutorul cărora procesul de dezvoltare să se realizeze într-o manieră cît mai simplă şi pe cît posibil, automatizată.

5.2. Etape ale metodologiei

Un sistem de control în timp real **S**, spre deosebire de sistemele informaționale clasice, este compus practic din două părți: procesul sau sistemul fizic de controlat **P** şi strategia de control utilizată **C**, adică:

$$S = P + C \qquad \text{(r5.1)}$$

Fiecare din aceste componente ale sistemului poate fi descrisă într-o manieră formală sau informală; maniera de descriere informală conduce la realizarea unui aşa numit model descriptiv (informal sau semiformal) al sistemului:

$$M_d = P_d + C_d, \text{ unde} \qquad \text{(r5.2)}$$

$\mathbf{M_d}$ = modelul descriptiv al sistemului S
$\mathbf{P_d}$ = modelul descriptiv al procesului de controlat P
$\mathbf{C_d}$ = modelul descriptiv pentru strategia de control C

In cazul utilizării descrierilor formale, rezultă aşa numitul model prescriptiv al sistemului:

$$M_p = P_p + C_p. \qquad \text{(r5.3)}$$

$\mathbf{M_p}$ = modelul prescriptiv al sistemului S
$\mathbf{P_p}$ = modelul prescriptiv al procesului de controlat P
$\mathbf{C_p}$ = modelul prescriptiv pentru strategia de control C

Astfel, în esență, procesului de proiectare a sistemului timp real trebuie să realizeze maparea modelului descriptiv $\mathbf{M_d}$ pe modelul prescriptiv $\mathbf{M_p}$, adică:

$$M_d => M_p \qquad \text{(r5.4)}$$

Modelul prescriptiv este testat utilizînd simularea; în urma validării acestuia prin simulare, se va trece apoi la construirea modelului de implementare a sistemului $\mathbf{M_{impl}}$, pe baza căruia va fi realizată ulterior implementarea propriu-zisă. Se va utiliza schema de dezvoltare iterativă, prezentată în cadrul paragraful 1.3.3. în procesul de proiectare şi respectiv de realizare a modelelor de mai sus. După cum a fost precizat deja, aceasta este bazată în principal pe binecunoscutul principiu al ingineriei programării *separation of concerns* şi al *feedback*-ului. Validarea modelului este posibilă deoarece maparea lui $\mathbf{M_p}$ pe $\mathbf{M_d}$, trebuie să fie combinată cu maparea lui $\mathbf{M_d}$ pe $\mathbf{M_p}$, pentru a realiza feedback-ul.

Referitor la modelele definite mai sus, pot fi enunțate următoarele proprietăți:

- dacă modelul descriptiv $\mathbf{M_d}$ este suficient de apropiat de sistemul **S** atunci concluziile rezultate pentru modelul $\mathbf{M_d}$ sunt valabile şi pentru sistemul **S**
- dacă sistemul **S** este construit pe baza modelului prescriptiv $\mathbf{M_p}$ atunci concluziile rezultate pentru modelul $\mathbf{M_p}$ sunt de asemenea valabile şi pentru sistemul **S**
- în general $\mathbf{M_d} \subset \mathbf{M_p}$; din acest motiv verificarea faptului că modelul prescriptiv $\mathbf{M_p}$ este în conformitate cu modelul descriptiv $\mathbf{M_d}$ este mai uşoară decît vice-versa.

Avantajele utilizării schemei de dezvoltare iterative în contextul metodologiei propuse sunt următoarele:

- aplicarea principiului ingineriei programării *separation of concerns*
- posibilitatea validării încrucişate a modelelor $\mathbf{M_p}$ si $\mathbf{M_d}$ datorită feedbak-ului

Ca dezavantaj, trebuie precizat că utilizarea variantei iterative este mai puțin eficientă din punctul de vedere al timpului de dezvoltare, rezultînd probabil într-un timp de dezvoltare ceva mai lung decît în cazul utilizării celorlalte variante (respectiv secvențială şi concurentă).

Figura 21 ilustrează într-o manieră schematică etapele metodologiei propuse. Se observă in cadrul acesteia utilizarea combinată a tehnicilor din domeniul **ICA** şi **IP** în cadrul procesului de dezvoltare. Astfel, în ceea ce priveşte domeniul **IP**, se observă că metodologia respectă toate etapele ciclului de viață al unui sistem de control timp real care au fost prezentate în cadrul paragrafului 1.3.4.

Dintre acestea, în cadrul etapei de control algoritmic, sunt utilizate pregnant tehnicile domeniului **ICA** pentru adoptarea strategiei de control şi analiza acesteia utilizînd simularea. Utilizarea feedback-ului pentru fiecare etapă asigură, după cum a fost precizat anterior, posibilitatea validării incrucişate a tuturor modelelor dezvoltate, conform schemei de dezvoltare iterativă prezentată în cadrul paragrafului 1.3.3.

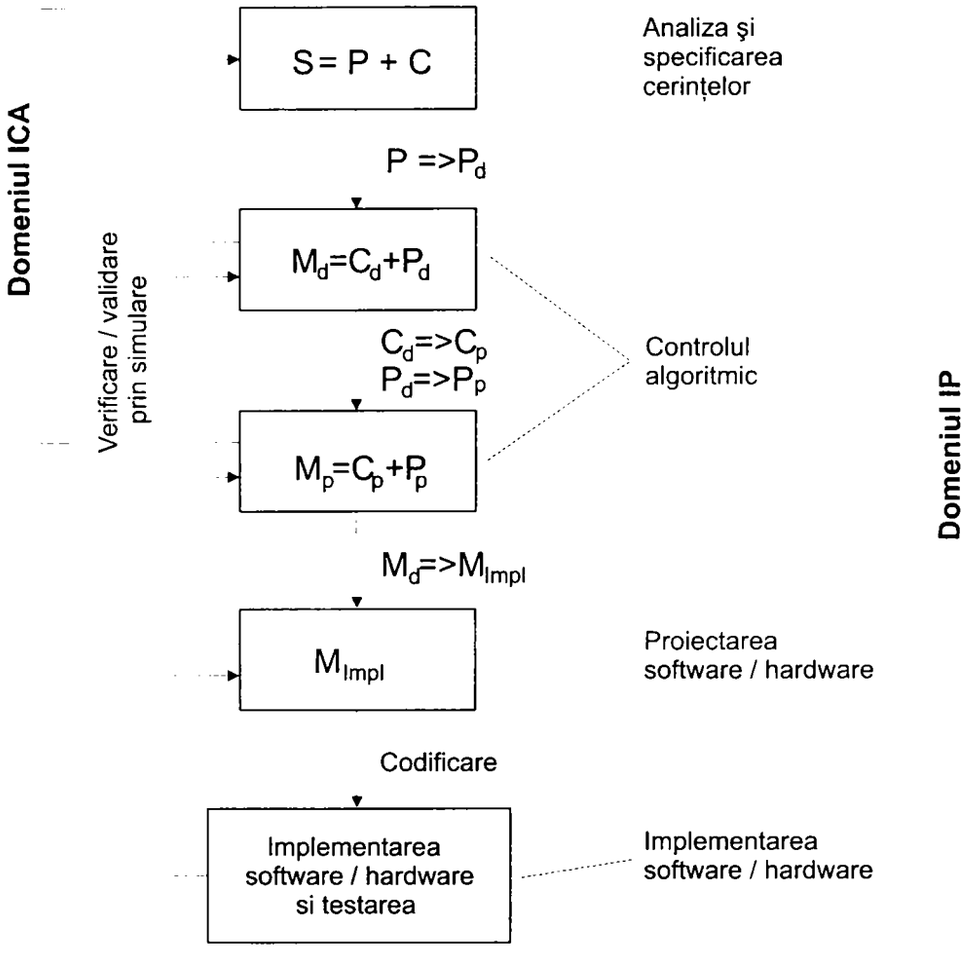


Figura 21: Etapele metodologiei propuse

O descriere detaliată a acestor etape este prezentată în continuare.

5.2.1. Etapa 1: Realizarea modelului descriptiv M_d

Scopul acestei etape îl reprezintă realizarea modelului descriptiv $\mathbf{M_d}$ al sistemului de control timp real, pornind de la documentul care conţine specificaţiile sistemului. Acest document trebuie să conţină în mod obligatoriu următoarele elemente:

- obiectivele sistemului
- o descriere detaliată a sistemului în ansamblul său
- specificarea intrărilor şi ieşirilor sistemului
- o descriere detaliată a modului în care sistemul în ansamblul său trebuie să reacționeze în diferite situații
- definirea cerințelor sistemului (constante, parametri, cerințe temporale, etc.) precum şi a modului în care acestea vor fi confirmate: inspecție, analiză sau test

Paşii prin intermediul cărora este construit modelul $\mathbf{M_d}$ sunt următorii:

5.2.1.1. Descompunerea sistemului în subsisteme

Metoda de descompunere utilizată are la bază descompunerea structurală bazată pe funcționalitate; tehnica de modelare adoptată fiind tehnica ***top-down***. Astfel, pe baza specificării cerințelor se analizează şi se determină care sunt principalele funcții ale sistemului de control în timp real. Se utilizează în această primă fază diagramele de scheme bloc ca şi unealtă de bază a modelării. Principalele elemente ale acestora sunt:

- blocurile care identifică subsistemele
- săgeți cu etichete care reprezintă mărimile de intrare/ieşire din blocul respectiv şi de asemenea stabilesc şi legăturile între diversele blocuri

Principalele criterii care trebuie luate în considerare în cadrul acestei descompuneri sunt următoarele:

- ***coeziunea funcțională***: anumite elemente pot fi grupate deoarece ele realizează un set de funcții legate între ele din punct de vedere funcțional
- ***coeziunea secvențială***: anumite elemente realizează funcții care trebuie să aibă loc într-o anumită ordine (secvențial)
- ***coeziunea temporală***: anumit elemente pot fi grupate pe criteriu că acestea sunt declanşate de un anumit eveniment particular; pentru sistemele în timp real gruparea pe baza coeziunii temporale este esențială: comportarea în timp a acestor transformări poate fi uşor urmărită şi analizată
- ***dependența de I/O*** : este utilă gruparea în module separate a elementelor care sunt puternic legate de I/O;

- ***element fizic sau de control***: anumite module sau blocuri se referă la un element fizic existent în cadrul sistemului pe cînd alte module au o natură mult mai abstractă, ele înglobînd practic strategia de control a anumitelor dispozitive; în scopul dezvoltării ulterioare cît mai rapide a programului de control, în cadrul fazei de implementare, este necesară utilizarea blocuri separate pentru fiecare categorie

Fiecare dintre subsistemele identificate sunt în continuare descompuse ierarhic, pînă la nivelul la care acestea ajung să identifice un element fizic din sistem, şi care nu mai poate fi descompus în continuare (de exemplu, o valvă). Pentru fiecare bloc astfel obținut trebuie specificate:

- numele asociat blocului
- marimile de intrare către blocul respectiv
- marimile de ieşire din blocul respectiv

Se utilizează în acest scop un ***dicționar de date*** pentru definirea tuturor mărimilor conținute în cadrul diagramelor bloc de mai sus. Acest dicționar va conține numele acestora, elementul fizic cu care mărimea este asociată, dacă este cazul, precum şi explicații pe scurt referitoare la mărimea în cauză. Descrierea obținută astfel reprezintă modelul descriptiv al procesului, $\mathbf{P_d}$.

5.2.1.2. Definirea strategiei de control

In continuare modelul obținut $\mathbf{P_d}$ se completează cu elementele de control (ex: buclele de reglare) necesare, rezultate în urma analizării strategiei de control descrise în faza de specificare. Acestea definesc practic strategia de control a sistemului, $\mathbf{C_d}$. Astfel, fiecărui element controlat i se asociază un bloc de control în cadrul diagramei bloc de mai sus, şi care va conține:

- mărimea controlată (ex: temp, presiune, debit, etc.)
- modalitatea prin care se realizează controlul (ex: inchiderea sau deschiderea a unei valve, oprirea/pornirea unui motor, etc.)
- diferiți parametri asociați controlului, determinați pe baza specificațiilor impuse

Prin completarea modelului $\mathbf{P_d}$ cu strategia de control rezultă modelul descriptiv semi-formal al sistemului $\mathbf{M_d}$, adică:

$$M_d = P_d + C_d, \qquad \text{(r5.5)}$$

unde $\mathbf{C_d}$ reprezintă modelul pentru strategia de control adoptată.

5.2.2. Etapa 2: Realizarea modelului prescriptiv formal M_p şi verificarea acestuia utilizînd simularea

Avînd în vedere schema de testare în patru paşi caracteristică ciclului de viață a sistemelor de control timp real (Figura 8), şi care presupune realizarea testării (verificării) în toate etapele ciclului de dezvoltare, este necesar ca următor pas în dezvoltarea sistemului de control timp real să se realizeze şi testarea modelului propus împreună cu strategia de control adoptată. Acest lucru se realizează în această fază prin simularea modelului obținut în urma fazei de descompunere, utilizînd o unealtă de simulare adecvată.

Pentru ca acest lucru să fie posibil, trebuie să fie îndeplinite două condiții esențiale:

- descrierea anterioară a modelului informal sau semiformal $\mathbf{M_d}$ să fie realizată utilizînd o unealtă CASE care să permită obținerea ulterioară a modelului prescriptiv formal $\mathbf{M_p}$ precum şi a simulării acestuia
- avînd în vedere complexitatea şi numărul mare a scenariilor de test existente în cazul sistemelor de control timp real, trebuie să existe posibilitatea generării automate a acestora sau a definirii unei metode care să uşureze în mod considerabil procesul de testare

Majoritatea proceselor fizice sunt la ora actuală modelate prin intermediul ecuațiilor diferențiale continue. Acest domeniu este la ora actuală deosebit de bine dezvoltat, atît din punct de vedere al matematicii care se află la baza acestor ecuații, cît şi din punct de vedere al uneltelor CASE existente pe piată pentru acest domeniu.

Majoritatea uneltelor CASE actualmente disponibile pe piață utilizează conceptul de **MMS (Modular Modeling System)** în sensul că există deja create biblioteci de componente care pot fi relativ uşor utilizate şi/sau adaptate conform cerințelor unui anumit model particular. Utilizarea unor astfel de biblioteci împreună cu un limbaj general de simulare furnizează un model simplu pentru ilustrarea dinamicii sistemului de controlat şi care conduce la o simulare simplă şi uşor de operat.

In cadrul acestei etape, pe baza modelului $\mathbf{M_d}$ inițial, se obține modelul prescriptiv $\mathbf{M_p}$ al sistemului de control (sau modelul matematic), prin specificarea la nivel de ecuații diferențiale a tuturor blocurilor de pe nivelurile de bază a ierarhiei obținute în cadrul modelului $\mathbf{M_d}$.

5.2.2.1. Realizarea modelului M_p şi simularea acestuia

In procesul de creare a modelului $\mathbf{M_p}$ şi de simulare a acestuia, trebuie să fie urmate următoarele faze:

- determinarea cerinţelor pe care simularea finală trebuie să le satisfacă => aceste cerinţe au fost specificate în cadrul modelului $\mathbf{M_d}$, care reprezintă în acest caz punctul de plecare; o categorie aparte în acest sens o reprezintă cerinţele temporale
- dezvoltarea ecuaţiilor care descriu sistemul precum şi a logicii de control acolo unde este cazul
- programarea acestor ecuaţii pe un sistem de calcul în scopul simulării ulterioare a acestora
- compararea programului care reprezintă simularea cu ecuaţiile sistemului şi modificarea simulării atunci cînd sunt decoperite greşeli (verificarea simulării)
- compararea rezultatelor simulării cu date experimentale obţinute din studierea sistemului şi modificarea modelului şi, corespunzător, a simulării dacă se constată neconcordanţe (validarea modelului matematic)
- acceptarea modelului (şi implicit a simulării acestuia) şi pregătirea documentatiei care va fi utilizată în fazele următoare ale dezvoltării sistemului de control în timp real, în special în faza de implementare

In cadrul fazelor specificate mai sus, componentele a patra şi a cincea sunt de o importanţă covîrşitoare, acestea referindu-se la procesul de verificare şi validare al simulării şi respectiv al modelului matematic, şi nu trebuie confundate între ele.

Astfel, ***verificarea*** reprezintă procesul prin care se determină dacă ecuaţiile au fost programate corect în cadrul programului care implementează simularea; ***validarea*** în schimb, determină dacă programul de simulare se comportă corect prin studierea tuturor aspectelor pertinente în acest sens. Astfel, dacă simularea reprezintă într-adevăr cu exactitate modelul matematic, adică verificarea a fost realizată cu succes, simularea devine transparentă în cadrul procesului de validare a modelului, şi rezultatele simulării pot fi comparate cu datele experimentale cu un mare grad de încredere.

In paralel cu paşii de mai sus, este esenţială în cadrul acestei etape crearea unei documentaţii intermediare care să descrie în amănunt modelul prescriptiv $\mathbf{M_p}$ generat. Acest document va conţine:

- toate ecuaţiile şi logica care definesc matematic sistemul în forma în care acestea au trecut de procesul de verificare şi validare

- identificarea şi justificarea tuturor presupunerilor utilizate în dezvoltarea modelului şi a simulării acestuia
- identificarea tuturor constrîngerilor de timp care trebuiesc satisfăcute şi a modalității în care acestea au fost incluse în cadrul modelului $\mathbf{M_p}$
- un *dicționar de date* complet a tuturor variabilelor şi constantelor utilizate, numele acestora şi ceea ce reprezintă fiecare variabilă sau constantă; acest dicționar are ca punct de pornire dicționarul de date dezvoltat în cadrul modelului $\mathbf{M_d}$, pe care în completează. In cadrul acestuia, sunt identificate acele variabile care nu sunt specifice doar procesului de simulare ci sunt asociate unor elemente care vor exista în sistemul fizic real: variabile asociate cu I/O, fanioane, variabile care conțin rezultate ale unor calcule numerice (totalizări, etc.), constante care identifică anumite limite impuse de specificații, constante PID, etc.

Astfel, procesul de creare a modelului $\mathbf{M_p}$ şi de simulare al acestuia poate fi descris prin următoarea diagramă:

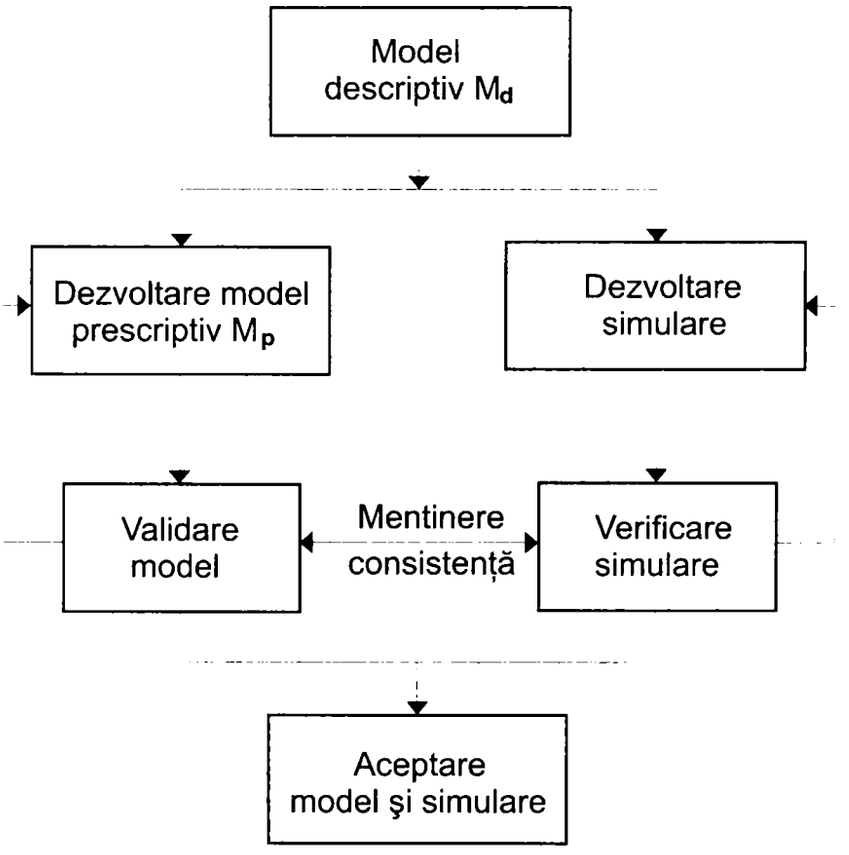


Figura 22: Paşi în dezvoltarea modelului şi a simulării

După cum este ilustrat în Figura 22, dezvoltarea modelului şi a simulării are loc în paralel (conform schemei generale de dezvoltare iterativă, Figura 6). Acest lucru implică unele probleme atunci cînd este necesar să se revină în urma descoperirii unor erori, în sensul că trebuie menținută permanent consistența între model şi simulare [Beq98].

5.2.2.2. Testarea modelului generat şi al simulării

Principala dificultate întîlnită în procesul de simulare în practică reprezintă faptul că procesul de validare este deosebit de complex datorită numărului scenariilor de test care creşte exponențial cu complexitatea sistemelor. Generarea automată a scenariilor de test şi deci automatizarea procesului de validare reprezintă un mare avantaj pe care multe din uneltele **CASE** existente la ora actuală îl pun la dispoziție. Chiar şi în această situație procesul de validare nu este garantat că se termină într-un timp finit, şi în consecință răspunsul de cele mai multe ori nu este da sau nu, ci în cele mai multe cazuri "poate".

Este însă important ca să existe o simulare verificată înaintea începerii procesului de validare. Sistemul real precum şi simularea verificată sunt apoi executate utilizînd scenarii de test identice, şi rezultatele sunt comparate. In urma acestui proces, pot rezulta schimbări atît în cadrul ecuațiilor cît şi în cadrul simulării.

Procesul se reia, pînă cînd ieşirile pentru cele două cazuri se încadrează în limitele de toleranță impuse. In cazul în care simularea nu a fost verificată în prealabil, nu va fi posibil să se determine dacă neconcordanțele apărute între datele de ieşire sunt cauzate de un model eronat sau de o simulare defectuoasă a acestuia.

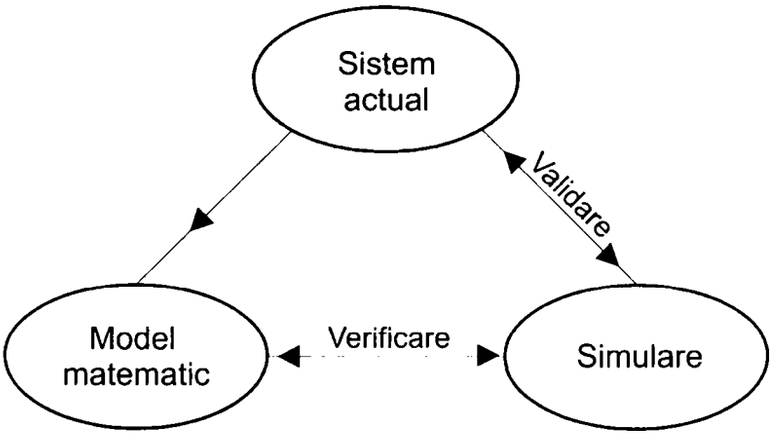


Figura 23: Relațiile între sistem, model, simulare, verificare şi validare

In consecință, există două tipuri de testări asociate cu modelarea şi simularea: ***testarea în scopul verificării*** şi ***testarea în scopul validării***. Testarea în scopul verificării implică testarea software-ului de simulare în scopul determinării dacă acesta reflectă modelul specificat. Testarea în scopul validării implică compararea rezultatelor simulării cu rezultatele experimentale obținute din cadrul sistemului real. Relațiile dintre sistemul fizic propriu zis, modelul matematic precum şi simularea acestuia sunt ilustrate în Figura 23. Practic, modelul prescriptiv $\mathbf{M_p}$ al sistemului reprezintă un prototip de simulator care serveşte în principal două scopuri distincte:

- verificarea specificațiilor sistemului, inclusiv a strategiei de control adoptate
- furnizarea unei imagini de ansamblu asupra dinamicii sistemului în general
- determinarea aproximativă proprietăților temporale ale acestuia.

Practic, se pot distinge două categorii de simulări care sunt utilizate pentru validarea proprietăților unui program de control în timp real***: simularea funcțională a sistemului*** şi ***simularea performanțelor sistemului***:

- ***simularea funcțională*** - are ca scop investigarea proprietăților de bază ale modelului utilizat şi a funcționării acestuia în conformitate cu specificațiile date, pe baza comparațiilor cu date culese din sistemul real. Rezultatele obținute astfel pot ajuta la analizarea şi depistarea unor eventuale erori în procesul de proiectare, în special a strategiei de control $\mathbf{C_p}$. In plus, prin utilizarea unor astfel de simulări pot fi furnizate anumite răspunsuri în fazele ulterioare de întreținere şi mai ales de dezvoltare a sistemului.
- ***simularea performanțelor sistemului*** – are ca scop evaluarea performanțelor sistemului, cu un anumit grad de încredere dat de acuratețea simulării. Acest tip de simulare este util în următoarele cazuri:
  - atunci cînd este necesară realizarea unor anumite măsurători în cadrul sistemului pentru un anumit grad de încărcare, simulatorul dă posibilitatea izolării proprietăților specifice în cadrul unui mediu specific; pentru sistemele în timp real în mod special, permite evaluarea performanțelor temporale ale acestora în situațiile de încărcare maximă (cele mai defavorabile). Trebuie aici specificat faptul că, constrîngerile temporale caracteristice acestor tipuri de sisteme (sisteme de monitorizare şi control în timp real) sunt derivate în principal nu atît din nişte deadline-uri caracteristice instrumentelor utilizate, întrucît dispozitivele utilizate operează în general periodic, ci mai degrabă din necesitatea de a asigura siguranța în funcționare a obiectelor controlate

- atunci cînd este necesară realizarea unor anumite testări pentru situațiile critice şi pentru care testarea reală prezintă pericole; în acesfel de cazuri testarea în timp real poate fi extrem de dificil de realizat, uneori chiar imposibilă

- poate fi utilizată ca şi o unealtă de depanare în sensul că, pe baza rezultatelor simulării se poate trece la reproiectarea anumitor părți ale sistemului în cazul în care s-a ajuns la situații de blocare

Procesul de evaluare al performanțelor este deosebit de complex, şi necesită o serie de etape:

- determinarea modalității efective în care va fi evaluată o anumită mărime, adică descrierea detaliată a scenariului de execuție utilizat pentru fiecare caz în parte
- în cazul evaluării cerințelor temporale, se vor realiza simulări utilizînd diferite ***intervale de simulare***, din ce în ce mai scurte; intervalul (cuanta) de simulare repezintă cuanta de timp după care programul reevaluează toate mărimile şi actualizează valorile acestora. Cuanta de timp pentru care performanțele obținute sunt satisfăcătoare va constitui punctul de pornire în alegerea tipului de PLC utilizat. Astfel, pentru programul care va fi implementat în cadrul PLC-ului va trebui să se asigure o perioadă de scanare dT_{max} mai mică decît cuanta de timp rezultată în urma simulării
- obținerea unei siguranțe în ceea ce priveşte rezultatele obținute prin rularea unui număr mare de simulări, cu diferiți parametri
- dezvoltarea unei metode sistematice de analiză care dă posibilitatea determinării dacă sistemul îndeplineşte cerințele de performanță impuse şi în caz că nu, identificarea situațiilor limită şi utilizarea acestora în procesul de modificare şi optimizare a sistemului modelat

Modalitatea practică de realizare a validării se bazează în principal pe stabilirea unei ierarhii de sisteme, subsisteme şi module care trebuie testate, precum şi a scenariilor de test utilizate pentru fiecare în parte. Fiecare scenariu trebuie să verifice în mod obligatoriu următoarele:

- cazurile de încărcarea maximă
- să determine experimental timpul ***aproximativ*** de execuție pentru toate activitățile cu un deadline ferm, în condițiile unei cuante de simulare date
- să testeze comportarea sistemului în condițiile simulării de defecțiuni hardware şi software

Aceste scenarii de test cresc în complexitate pe măsură ce se trece de la testarea individuală, la nivel de modul şi subsistem, către testarea sistemului în ansamblul său (testarea de integrare). Uneori testarea finală, integrare, este deosebit de dificil de realizat în cazul unor sisteme de o complexitate mare, şi, pentru aceste cazuri trebuiesc utilizate metode de testare puțin diferite de cele clasice. O astfel de metodă de testare o reprezintă ***testarea interactivă***.

Definiție:
Testarea interactivă – reprezintă o metodă de testare a unui sistem de control timp real utilizînd un singur scenariu de test, dar a cărui parametri pot fi modificați interactiv in decursul simulării

O comparație între modalitățile obişnuite de testare şi testarea interactivă este prezentată în Figura 24:

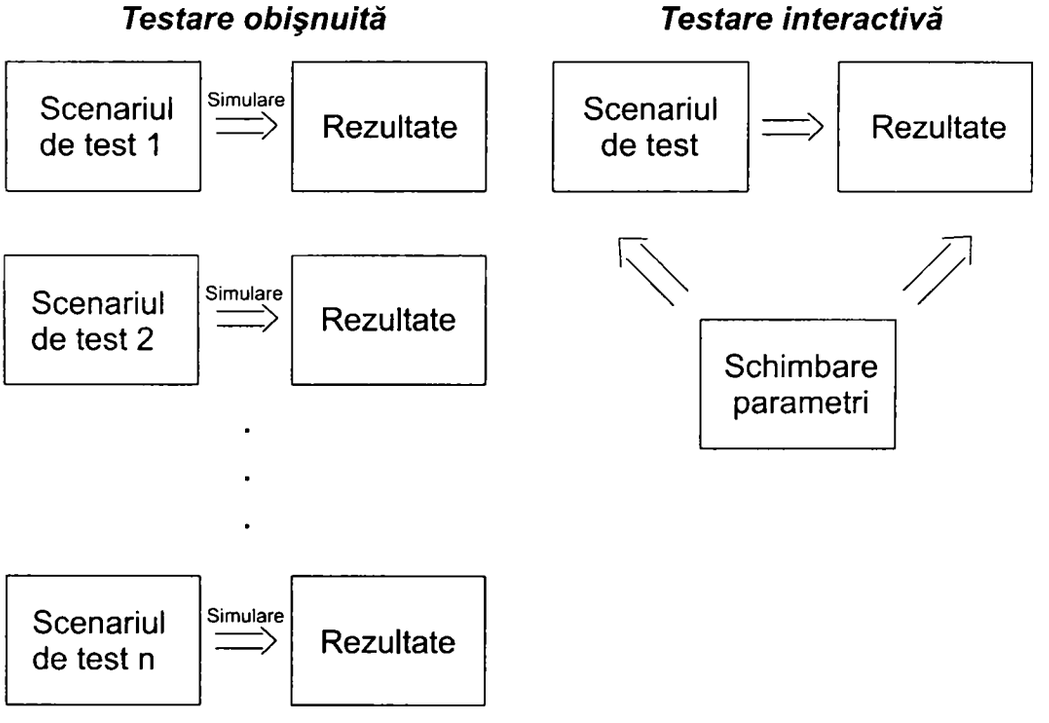


Figura 24: Comparație între testarea obişnuită şi testarea interactivă

Pentru ca testarea interactivă să fie realizată practic trebuie să existe posibilitatea modificării interactive a parametrilor utilizați în cadrul unui scenariu de test. In cazul sistemelor de control în timp real a proceselor industriale, acest lucru se poate realiza, dacă se dispune de uneltele CASE adecvate, utilizînd chiar interfața grafică asociată procesului controlat.

Conform criteriilor de descompunere în subsisteme prezentate în paragraful 3.2.1, aceasta va rula pe un nod separat celui (sau celor) pe care va rula programul de control, lucru dictat de raţiuni legate de o mai bună structurare precum şi de necesitatea de a minimiza pe cît posibil dimensiunea programului de control propriu-zis. În plus, toate prelucrările şi analizele ulterioare de ordin statistic (trasare de diagrame în timp real, stocarea datelor şi trasarea diagramelor de evoluţie în timp a diverşilor parametri, etc.) nu vor fi incluse în programul de control, ci vor fi parte a interfeţei utilizator.

Marele avantaj al utilizării metodei de testare interactivă faţă de metodele clasice este flexibilitatea: în cadrul acesteia, diferite scenarii de test pot fi generate în decursul execuţiei unei singure simulări prin schimbarea interactivă a unor parametri din cadrul scenariului iniţial; în cadrul metodelor de testare clasice, după execuţia fiecărui scenariu, se reia simularea pentru următorul scenariu de execuţie, ş.a.m.d., pînă la epuizarea tuturor scenariilor posibile (Figura 24).

Intrucît posibilitatea de a realiza o astfel de modalitate de testare este strîns legată de disponibilitatea unui mediu software pentru dezvoltarea interfeţelor utilizator care să permită realizarea legăturii cu simulatorul, unelte CASE adecvate trebuie alese în acest scop. In consecinţă, software-ul utilizat pentru dezvoltarea interfeţei utilizator trebuie să furnizeze trei elemente esenţiale:

- un editor grafic care să dispună de biblioteci de componente cît mai diversificate şi care să permită descrierea la nivel schematic a sistemului într-o manieră cît mai sugestivă
- metode de acces la baza de date utilizată în procesul de simulare şi posibilitatea modificării interactive acesteia (stabilirea legăturii bidirecţionale cu simularea)
- posibilitatea realizării unor activităţi de managementul datelor: stocarea şi întreţinerea datelor pentru prelucrări ulterioare, realizarea de grafice de evoluţie în timp real, precum şi a altor calcule necesare la nivelul aplicaţiei

Utilizînd o astfel de interfaţă, modelul prescriptiv simulat $\mathbf{M_p}$ se comportă în această fază ca şi sistemul controlat real, transmiţînd date către interfaţa utilizator şi primind date şi comenzi prin intermediul acesteia. Pentru un anumit scenariu, utilizatorul poate schimba diferiţi parametri de rulare pentru a studia efectele acestora în sistem. Astfel se poate obţine o imagine mult mai fidelă a dinamicii sistemului decît în cazul în care interfaţa cu utilizatorul nu ar fi fost utilizată, datorită faptului că, prin intermediul acesteia, interpretarea rezultatelor devine mult mai clară.

5.2.3. Etapa 3: Construirea modelului pentru implementare utilizînd PLC-uri

Scopul acestei etape îl constituie realizarea modelului de implementare $\mathbf{M}_{impl}$ a sistemului de control timp real pe baza modelului prescriptiv validat $\mathbf{M}_p$ obținut în cadrul etapei anterioare. Construirea modelului de implementare $\mathbf{M}_{impl}$ constă practic din două etape esențiale: împărțirea software-ului în module (proiectarea software) si alocarea acestora pe procesoare (proiectarea hardware). Modalitatea efectivă în care se realizează acest lucru este dependentă de rezultatele obținute în procesul de validare al modelului prescriptiv $\mathbf{M}_p$.

5.2.3.1. Proiectarea software

Practic, strategia general recomandată poate fi exprimată ca fiind bazată pe minimizarea pe cît posibil a părților din sistem care au constrîngeri de timp hard, pentru că acestea sunt cel mai dificil de implementat şi testat. In consecință, structura programului trebuie să se bazeze pe partiționarea aplicației în module pe baza funcționalității şi a constrîngerilor de timp, în cazul în care este necesar, şi care sunt cuplate împreună utilizînd o structură de control, ca cea prezentată în Figura 19. Această partiționare conduce la un set de operații, fiecare operație realizînd o funcție specifică. Aceste operații realizează de fapt controlul procesului: adică transformările necesare asupra datelor preluate de la senzori şi transmise către actuatori.

In procesul de generare a modelului pentru implementare $\mathbf{M}_{impl}$ şi ulterior a implementării acestuia, ideea de bază o reprezintă păstrarea structurii codului din cadrul modelului $\mathbf{M}_p$ care corespunde software-ului ce va fi implementat în controller. Astfel, în general, definițiile care compun strategia de control $\mathbf{C}_p$ din cadrul modelului $\mathbf{M}_p$ constă din blocuri multiple interconectate între ele prin intermediul unei structuri ierarhice.

Această organizare ierarhică este benefică din mai multe puncte de vedere, printre care:

- ***economia de expresii*** – componente comune pot fi referite în mod repetat fără a duplica definiția acestora de fiecare dată cînd sunt utilizate
- ***controlul asupra granularității definițiilor în diferite contexte*** – anumite definiții complexe pot fi sparte în blocuri mai mici, de dimensiuni mai uşor de asimilat
- ***reutilizarea şi interschimbarea subcomponentelor*** – acestea pot fi uşor actualizate şi împărțite între definiții de control multiple

- ***costuri de întreținere scăzute*** - componentele comune pot fi întreținute şi reutilizate prin intermediul unei biblioteci centrale de componente

In urma validării modelului simulat $\mathbf{M_p}$ al sistemului se poate trece la construirea modelului utilizat pentru implementarea propriu-zisă. Construirea modelului de implementare $\mathbf{M_{impl}}$ implică extragerea din cadrul modelului $\mathbf{M_p}$ a tuturor informațiilor necesare în implementarea programului de control. Aceste informații sunt următoarele:

- ***strategia de control*** $\mathbf{C_p}$ care va fi implementată în cadrul programului de control: fiecare bloc de control din cadrul modelului $\mathbf{M_p}$ va fi transpus într-un bloc de program corespunzător blocului respectiv
- ***dicționarul de date***, care conține toate variabilele utilizate în cadrul modelului: din cadrul acesteia se vor extrage doar variabilele care sunt utilizate în cadrul strategiei de control precum şi variabilele care se referă la I/O. In scopul automatizării pe cît posibil a activității de construire a modelului şi implementare a acestuia, va trebui realizată o corespondență directă între aceste variabile şi variabilele utilizate în acelaşi scop din cadrul PLC-ului. Prin înlocuirea variabilelor modelului $\mathbf{M_p}$ cu variabilele corespunzătoare din PLC în cadrul strategiei de control $\mathbf{C_p}$ de mai sus, se obține modelul general de implementare al programului de control $\mathbf{M_{impl}}$

Practic acest lucru se realizează prin utilizarea sursei programului de simulare propriu-zis, cu următoarele modificări:

- înlăturarea porțiunilor care se referă la modelarea elementelor fizice din sistem şi care nu țin de logica de control
- realizarea corespondenței variabilelor din dicționarul de date care se referă la I/O şi care sunt utilizate în cadrul strategiei de control cu variabilele utilizate în cadrul PLC-urilor
- păstrarea structurii generale a modulelor şi eventual completarea acestuia cu noi module

Atunci cînd logica de control este definită într-o manieră ierarhică, ca şi în cazul modelului $\mathbf{M_p}$, în procesul construirii modelui de implementare $\mathbf{M_{impl}}$ este ideală păstrarea acestei structuri, modulele din cadrul ierarhiei vor putea fi astfel mapate unu-la-unu peste modulele (subrutine, blocuri) din cadrul software-ului de control. Acest lucru este deosebit de important deoarece, modelul de implementare $\mathbf{M_{impl}}$ fiind derivat din cadrul modelului prescriptiv $\mathbf{M_p}$, care a fost în prealabil validat, este necesară testarea doar a porțiunii din cadrul modelului de implementare $\mathbf{M_{impl}}$ care corespunde unor module noi.

Aceasta arată faptul că, utilizarea unor biblioteci de componente reutilizabile verificate şi validate în cadrul strategiei de control $\mathbf{C_p}$, va conduce la crearea unor biblioteci de module (subrutine) software reutilizabile şi certificate în cadrul software-ului implementat. Utilizarea unei astfel de abordări în realizarea modelului de implementare $\mathbf{M_{impl}}$ are următoarele avantaje:

- modelul de implementare păstrează structura ierarhică a modelului $\mathbf{M_p}$ simulat; în consecinţă şi programul implementat după acest model va fi structurat ierarhic într-o manieră asemănătoare
- modelul $\mathbf{M_p}$ fiind descris într-o manieră formală, rezultă că şi modelul de implementare $\mathbf{M_{impl}}$ rezultat va fi descris tot formal; în consecinţă, transpunerea acestuia într-o implementare utilizînd un limbaj specific pentru un anumit tip de controller este foarte simplă, clară şi directă. Această translatare poate fi eventual şi automatizată prin crearea unor translatoare specifice.

5.2.3.2. Alocarea modulelor pe procesoare

Activitatea de alocare necesită analiza întregului sistem pentru estimarea performanţelor acestuia în relaţie cu anumite constrîngeri de timp impuse de specificaţii. O astfel de analiză a fost realizată în cadrul procesului de simulare, şi ea a inclus următoarele etape:

- determinarea experimentală a timpilor de execuţie pentru toate acţiunile cu constrîngeri de timp hard, în condiţiile simulării realizate cu diverse intervale de simulare
- compararea timpilor obţinuţi cu cei impuşi de specificaţii

Astfel, în funcţie de rezultatele obţinute, există două posibilităţi de abordare:

- ***utilizarea unui singur procesor*** – reprezintă cea mai simplă modalitate de alocare şi ea presupune să se aloce un singur procesor pentru întreaga activitate de control. Acest lucru este posibil dacă, în urma analizei de mai sus, rezultă un t_lim_{min} astfel încît condiţia (r4.5) prezentată la paragraful 4.2.4. este îndeplinită pentru toate constrîngerile temporale din sistem în condiţiile tipului de PLC ales. Alegerea tipului de PLC se bazează pe caracteristicile acestuia, astfel încît sa poată fi asigurate resursele necesare conform ***principiului resource adequacy***: puterea procesorului, dimensiunea memoriei, posibilitatea de a asigura interfaţa cu dispozitivele, fiabilitatea. Alte cerinţe mai speciale care stau la baza alegerii sunt necesitatea unui set extins de instrucţiuni, cerinţe speciale de memorie, facilităţi de interfaţare cu dispozitive speciale, etc.

- ***utilizarea mai multor procesoare în cadrul unui sistem distribuit*** – în acest caz activitatea de control trebuie împărțită pe mai multe procesoare, astfel încît, pentru fiecare dintre acestea să poată fi îndeplinite condițiile (r4.5) şi eventual (r4.6) pentru toate activitățile cu deadline ferm. In acest scop, trebuie identificată structura sistemului şi modalitățile de comunicare între nodurile acestuia.

Pentru comunicarea între noduri în cazul sistemelor de control în timp real este de preferat să se utilizeze un server, central sau distribuit. O caracteristică a sistemelor de control în timp real care utilizează PLC-uri este aceea că achiziția de date se realizează periodic, şi deci datele sunt actualizate de asemenea periodic; această categorie de sisteme utilizează abordarea ciclică prezentată la paragraful 4.2.1. In această situație este mult mai eficient să se stocheze datele chiar la sursa colectării acestora cu posibilitatea de a fi accesate eventual de la distanță. De fapt, acest lucru este realizat implicit în cazul în care se utilizează abordarea **Cy-Clone distribuit**: după cum a fost precizat în paragraful 4.2.2., datele din cadrul tabelelor de comunicare ale fiecărui nod sunt accesibile tuturor celorlalte noduri.

Următoarele elemente trebuie luate în considerare în procesul de identificare a nodurilor:

- ***funcționalitate*** - nodul realizează o funcție sau un grup de funcții corelate; traficul de date între aceste funcții poate fi relativ mare şi împărțirea acestuia între mai multe noduri poate în mod potențial creşte prea mult traficul-ul din sistem; acțiuni de tipul ***A implică B*** vor fi pe cît posibil grupate pe acelaşi procesor, conform celor precizate in paragraful 4.2.4.
- ***server*** - nodul furnizează un serviciu, răspunzînd la cereri din partea unor alte noduri clienți; în mod frecvent server-ul furnizează servicii care sunt asociate cu un set de date
- ***agent*** - un nod agent furnizează un serviciu indirect; în scopul realizării acestui serviciu acesta trebuie să apeleze la alte subsisteme, acționînd ca un intermediar între un client şi un server
- ***controlul localizat*** - în anumite cazuri, nodurile furnizează anumite funcții specifice doar pentru o anumită parte a sistemului în ansamblul său. Astfel, se poate considera că nodul operează cu o relativă independență față de celelalte noduri, astfel el putînd fi operațional chiar şi în cazul în care unul dintre acestea se defectează

- ***interfața cu utilizatorul*** - odată cu proliferarea calculatoarelor personale, subsistemul care furnizează interfața cu utilizatorul este bine să ruleze pe un nod separat, interacționînd cu alte noduri. Practic, subsistemul pentru interfața cu utilizatorul are rol de actor [Booch]
- ***performanță*** – grupînd funcțiile critice din punct de vedere temporal în cadrul unui nod separat din sistem, de multe ori se poate obține o mai bună şi mai predictibilă performanță a acestuia

In plus, mesajele care implementează comunicarea între noduri pot fi tratate în două moduri:

- utilizînd o coadă de mesaje ***slab cuplată*** **(loosely coupled)**: dacă un task producător necesită transmiterea de informație către un task consumator şi cele două task-uri pot rula la viteze diferite
- utilizînd o coadă de mesaje ***puternic cuplată*** **(strong coupled)**: producătorul nu poate trece mai departe pînă cînd nu obține un răspuns de la consumator

Din punctul de vedere al modulelor software, în cazul în care se impune utilizarea mai multor procesoare în vederea satisfacerii anumitor cerințe temporale, în plus, față de etapele de translatare a modelului $\mathbf{M_p}$ în modelul $\mathbf{M_{impl}}$ descrise în cadrul capitolului anterior, mai este necesară în plus realizarea unei etape suplimentare de împărțire a modulelor modelului $\mathbf{M_{impl}}$ pe procesoarele utilizate, rezultînd un sistem distribuit, cu o arhitectură care se încadrează în modelul general prezentat în paragraful 4.1.2.

Criteriile fundamentale care stau la baza procesului de împărțire sunt în principal următoarele:

- gruparea, pe acelaşi procesor al modulelor cu coeziune mare, şi care realizează un set de funcții legate între ele astfel încît să se minimizeze cît mai mult posibil traficul de informație în sistem; utilizarea unei rețele de control în timp real (paragraful 4.1.2) pentru legarea între ele a diferitelor tipuri de PLC-uri pentru transmiterea în timp real a mesajelor este esențială în scopul obținerii de întîrzieri minime atît în procesul de transmitere al informațiilor (sincronizare la nivelul microsecundelor) cît şi în procesul de detectare a defecțiunilor unui anumit procesor
- întrucît în cadrul fiecărui PLC se utilizează abordarea ciclică, şi deci secvențială, în procesul de implementare propriu-zis, împărțirea modulelor trebuie astfel realizată încît, pentru fiecare procesor să se garanteze existența resurselor necesare astfel încît toate operațiile din cadrul programului să poată fi executate în timp util. Acest lucru

reprezintă de fapt premiza de bază, de la care se porneşte în cazul în care se doreşte utilizarea unei astfel de abordări (paragraful 4.2.).

Prin aplicarea acestei modalităti de descompunere şi mapare, rezultă în final o structură generală a software-ului timp real care reflectă cerinţele funcţionale ale sistemului de control în timp real, şi care conţine următoarele tipuri de module:

- module pentru realizarea controlului operational
- module pentru monitorizarea starilor şi realizarea achiziţiei de date
- module pentru tratarea defecţiunilor şi semnalarea alarmelor
- module pentru realizarea diferitelor calcule statistice
- module pentru analiza istoricului evoluţiei
- module pentru realizarea interfeţei cu utilizatorul

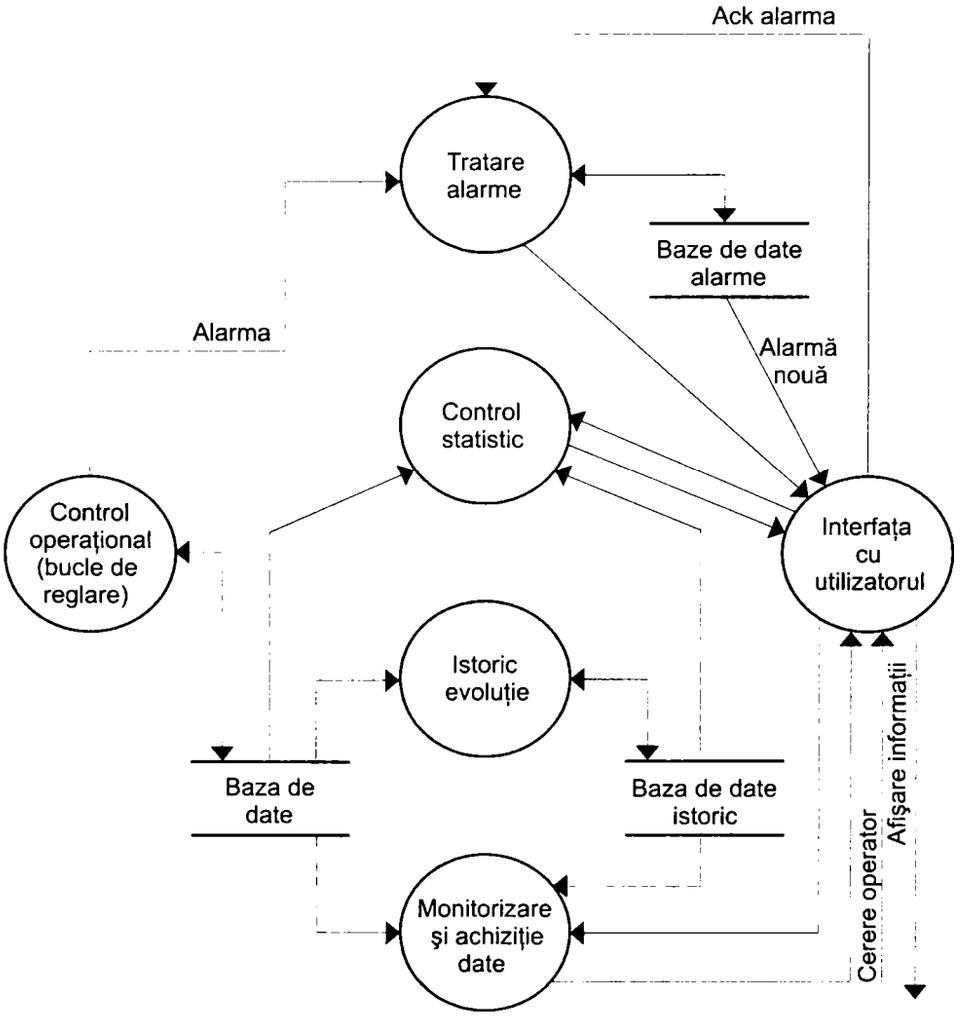


Figura 25: Tipuri conceptuale de module în sistemele de control timp real

Aceste categorii de module (Figura 25) sunt caracteristice oricărei categorii de aplicații de control automat în timp real, şi se vor regăsi ulterior şi în structura programului propriu-zis.

5.2.4. Etapa 4: Implementarea software-ului şi testarea

Scopul acestei etape îl constituie realizarea efectivă a implementării software-ului pentru sistemul de control timp real, pe baza modelului de implementare $\mathbf{M}_{impl}$ definit în cadrul etapei anterioare. Testarea software-ului implementat reprezintă ultima fază a procesului de testare care s-a desfăsurat de-a lungul întregului ciclu de dezvoltare a aplicației, conform schemei generale prezentată în cadrul paragrafului 1.3.4.5; în urma acesteia se va determina dacă într-adevăr sistemul proiectat şi implementat satisface cerințele funcționale şi temporale impuse.

5.2.4.1. Implementarea software-ului sistemului de control timp real

Implementarea software-ului de control se realizează pe baza modelului de implementare $\mathbf{M}_{impl}$ definit mai sus, utilizînd limbajul de programare specific controller-ului (sau controller-elor) ales. Implementarea poate fi realizată manual, sau automat, utilizînd în acest scop translatoare specifice. Deoarece modelul de la care se pleacă $\mathbf{M}_{impl}$ este derivat din $\mathbf{M}_{p}$, care este descris într-o manieră formală, transpunerea automatizată a acestuia într-o implementare nu reprezintă o problemă.

Generarea automată a codului este o opțiune care, în cazul în care este disponibilă este recomandat a fi utilizată, cu următoarele observații care trebuie avute în considerare:

- în ce măsură software-ul generat este sau nu optimizat – în cazul aplicațiilor de control în timp real este importantă realizarea unor optimizări chiar în cadrul procesului de translație. Aceast lucru reduce dependența de optimizările cu caracter general pe care le realizează divesele compilatoare utilizate şi de asemenea permite realizarea unor optimizări specifice domeniului
- în ce măsură software-ul generat este portabil pe calculatorul sau calculatoarele utilizate efectiv în controlul procesului
- în ce măsură codul generat poate fi accesat de alte aplicații astfel încît diverse forme de analiză să poată fi relizate în afara software-ului de control

In ciuda importanței păstrării structurii ierarhice, majoritatea generatoarelor automate de cod comerciale pentru controller-e existente la ora actuală nu păstrează structura definițiilor care implementează logica de control în software-ul generat. In mod tipic, subcomponentele din cadrul definițiilor sunt ca nişte macro-uri a căror segment de cod este substituit inline în cadrul software-ului generat. In consecință, software-ul produs nu prezintă modularitatea şi ierarhizarea modelului inițial; de cele mai multe ori acesta rezultă într-o subrutină mare şi unică, şi acest lucru impietează în mod hotărîtor asupra procesului de verificare şi validare, precum asupra întreținerii ulterioare a acestuia.

De asemenea, nu sunt tratate probleme legate de optimizarea codului în timpul procesului de translație; atunci cînd codul este destinat controller-elor care operează în timp real, realizarea unui cod cît mai compact, fiabil şi uşor de întreținut reprezintă elemente esențiale.

5.2.4.2. Testarea software-ului implementat

Procesul de testare se bazează pe ideea de a partiționa sistemul în subsisteme compozabile, astfel încît fiecare subsistem să poată fi testat în mod izolat, fără a fi generate efecte secundare în procesul de testare de integrare. Impărțirea software-ului pe subsisteme şi module conform criteriilor prezentate în paragraful 3.2.1. a avut în vedere acest lucru încă din faza de proiectare (proiectare pentru testabilitate);

Testarea software-ului implementat reprezintă ultima fază a procesului de testare care s-a desfăşurat de-a lungul întregii dezvoltări a aplicației de control în timp real (Figura 8, paragraful 1.3.4.5). Este practic singura fază care se realizează efectiv în timp real, programul fiind conectat la procesul fizic controlat. Scopul testării finale îl reprezintă acela de a se determina dacă sistemul îndeplineşte într-adevăr toate cerințele funcționale şi temporale.

In cadrul acestui proces de testare, observabilitatea ieşirilor subsistemelor existente precum şi controlabilitatea intrărilor utilizate reprezintă esența activității de testare. Utilizarea abordării ciclice în procesul de proiectare (sistem **time-triggered TT**) rezultă într-un sistem deterministic în cadrul căruia controlabilitatea intrărilor şi observabilitatea ieşirilor este relativ uşor de obținut, şi, în plus, reproductibilă.

Pentru sistemele care nu sunt de timp real, observabilitatea şi controlabilitatea sunt furnizate de către monitoare şi depanatoare de test care opresc programul în punctele de test şi dau posibilitatea vizualizării şi eventual

schimbării valorilor variabilelor. Pentru sistemele în timp real, o astfel de metodă nu este potrivită din două motive:

- întîrzierile temporale introduse la punctele de testare modifică comportarea temporală a sistemului astfel încît există posibilitatea ca anumite erori existente să fie ascunse
- în cazul unui sistem distribuit, există mai multe căi de control; oprirea uneia sau alteia poate conduce la distorsiuni temporale în sistem şi genera noi erori

De asemenea, de-a lungul activităţii de testare, numai o mică fracţiune din spaţiul intrărilor posibile al unui sistem software poate fi încercată. Astfel, un element esenţial în cadrul acestui proces îl reprezintă găsirea unui set cît mai reprezentativ de date pentru testare. Scenariile de utilizate în procesul de verificare şi validare a modelului prescriptiv $\mathbf{M_p}$ (simulatorului), pot fi utilizate şi în această fază.

Etapele care trebuie urmate în procesul de testare sunt următoarele:

- ***testarea separată a modulelor*** prin simularea prin program a fiecărei situaţii critice, în mod izolat, prin forţarea intrărilor corespunzătoare; în cadrul acestei faze programul nu a fost încă conectat la procesul fizic
- ***testarea programului în ansamblul său***, cu intrările şi ieşirile legate direct la procesul fizic de controlat

Pentru a uşura procesul de testare al sistemului se utilizează şi în această ultimă fază testarea interactivă prezentată în cadrul paragrafului 5.2.2.2. În această etapă însă, în locul prototipului de simulator se va afla sistemul real. Testarea a fost realizată intensiv, pe perioade lungi de timp, pentru a se putea culege suficient de multe date, şi pentru a se putea garanta, în proporţie cît mai apropiată de 100% funcţionarea corectă a sistemului din toate punctele de vedere. Practic, în această etapă a fost testat sistemul în ansamblul său, conectat direct la procesul fizic de controlat.

Această fază a testării în timp real a sistemului ridică în general probleme deosebite legate de protecţie, mai ales în cazul testării condiţiilor limită. In anumite cazuri, anumite situaţii nici nu pot fi testate "pe viu", luîndu-se în considerare în acest caz rezultatele obţinute în cadrul simulării. In consecinţă, acceptarea software-ului implementat ca fiind corespunzător, pe baza rezultatelor obţinute în procesul de testare, poate fi, în asemenea situaţii, discutabilă. Exceptînd însă astfel de situaţii, sistemul rezultat este un sistem deterministic, în cadrul căruia se poate garanta în final îndeplinirea tuturor constrîngerilor impuse.

5.3. Concluzii

In acest capitol a fost prezentată o metodologie de dezvoltare a aplicațiilor de control timp real utilizînd PLC-uri. Metodologia se bazează pe abordarea ciclică prezentată în cadrul paragrafului 4.2., este aplicabilă sistemelor care utilizează controlul temporal **(time triggered TT)**, şi constă în principal din următoarele etape:

- realizarea modelului descriptiv $\mathbf{M_d}$ pe baza specificațiilor inițiale ale sistemului; modelul $\mathbf{M_d}$ include şi strategia de control adoptată
- realizarea modelului prescriptiv $\mathbf{M_p}$ şi validarea acestuia utilizînd simularea
- realizarea modelului de implementare $\mathbf{M_{impl}}$ pe baza modelului $\mathbf{M_p}$ în prealabil validat
- implementarea software şi hardware a sistemului pornind de la modelul de implementare $\mathbf{M_{impl}}$ şi testarea acestuia

Fiind aplicată într-un domeniu multidisciplinar, metodologia se bazează pe utilizarea combinată a ***conceptelor ingineriei programării*** **IP** precum şi *a* ***ingineriei controlului automat*** **ICA**. Un element esențial în cadrul metodologiei propuse îl constituie utilizarea simulării în scopul validării modelelor propuse.

In acest sens, metodologia se bazează în principal pe combinarea a trei ingrediente de bază, şi anume:

- utilizarea unei unelte CASE pentru reprezentarea matematică **MMS** a componentelor sistemului în cadrul modelelor $\mathbf{M_d}$ şi $\mathbf{M_p}$
- utilizarea unui limbaj de simulare general pentru implementarea şi ecuațiilor pentru modelul matematic $\mathbf{M_p}$ descris mai sus
- utilizarea unui software de realizare a interfeței utilizator care să permită gestionarea în timp real a execuțiilor precum şi crearea unei interfețe utilizator interactive

Exemple de astfel de unelte **CASE** şi software de aplicație sunt multiple; o combinație care a fost utilizată şi care va fi prezentată în studiul de caz din cadrul capitolului 4 este următoarea:

- utilizarea mediului **ACSL /GM(Advanced Continuous Simulation Language / Graphic Modeller)** pentru reprezentarea **MMS** a componentelor sistemului
- utilizarea limbajului de simulare **ACSL** pentru simularea modelului matematic

- utilizarea software-ului de aplicație **InTouch WindowsViewer** pentru crearea interfeței utilizator interactive

Utilizarea simulării în combinație cu testarea interactivă permite evaluarea sistemului de control real implementat din toate punctele de vedere: funcțional, al performanțelor (şi în mod deosebit a celor temporale), al modului în care acesta reacționează în situațiile de defecțiune şi eroare, etc. In funcție de rezultatele obținute, vor fi luate în continuare deciziile referitoare la arhitectura hardware şi software utilizată în procesul de implementare propriu-zis, pe baza principiului de ***resource adequacy*** prezentat în paragraful 2.1.4.

Pentru sistemele timp real dezvoltate utilizînd metodologia propusă, garantarea performanțelor în toate situațiile poate fi realizată, datorită predictabilității obținute în urma utilizării paradigmei **Cy-Clone**. Problema este însă că metodologia nu este aplicabilă oricărui tip de sistem, ci doar acelora pentru care, pe baza constrîngerilor de timp impuse, este posibilă găsirea unei soluții hardware şi software astfel încît relațiile (r4.5) şi respectiv (r4.6), prezentate în paragraful 4.2.4., să poată fi îndeplinite. In prezent însă, odată cu dezvoltarea tot mai rapidă a puterii de procesare a controller-elor actuale, categoria de aplicații care pot fi modelate cu metodologia propusă este din ce în ce mai mare, reprezentînd marea majoritate a sistemelor întîlnite în practică.

Astfel, prin utilizarea combinată a PLC-urilor şi a paradigmei **Cy-Clone**, rezultă o metodologie de modelare şi simulare nouă şi importantă, care poate fi uşor adaptată pentru o largă categorie de aplicații complexe din domeniul controlului automat în timp real. Utilizarea concretă a metodologiei propuse va fi ilustrată pe larg în studiul de caz prezentat în cadrul capitolului 6.

6. STUDIU DE CAZ: SISTEMUL AUTOMAT DE MONITORIZARE ŞI CONTROL A INSTALAȚIEI GEOTERMALE DE LA UNIVERSITATEA DIN ORADEA

Pentru a furniza un exemplu concret de utilizare a metodologiei propuse în proiectarea şi implementarea unui sistem în timp real, se prezintă în continuare cazul concret al sistemului de monitorizare şi control a instalației geotermale de la Universitatea din Oradea, proiect dezvoltat de un colectiv din cadrul căruia am făcut parte.

Astfel, întreg campusul universitar orădean, precum şi cîteva clădiri din apropiere, utilizează, pentru încălzire şi apă caldă menajeră, apa geotermală extrasă dintr-un puț (nr. 4796) aflat în aria campusului. Pînă în anul 1997, potențialul acestuia nu a fost utilizat la capacitatea maximă, cauzînd astfel o serie de pierderi importante de energie.

În anul 1995, pe baza unui contract, şi în colaborare cu firma Raffhonnun din Reikjavik, Islanda, a fost demarat un proiect comun pentru modernizarea sistemului în cauză. O parte din modificările aduse vechiului sistem au constat în implementarea unui sistem automat de monitorizare şi control, care, față de versiunea complet manuală existentă pînă atunci, aduce clar o serie de avantaje:

- funcționare mult mai economică, optimă funcție de cerințe
- funcționare mult mai sigură
- monitorizarea continuă în timp a stării procesului, rezultînd reacții foarte rapide la eventuale situații de alarmă
- utilizarea unui minim de forță de muncă.

In procesul de dezvoltare a aplicației în timp real prezentate în cadrul acestui studiu de caz au fost urmate toate etapele prevăzute în cadrul metodologiei prezentate în cadrul capitolului 5. Capitolele care urmează ilustrează modalitatea în care metodologia propusă a fost în mod concret aplicată în procesul de proiectare şi implementare a sistemului de monitorizare şi control a instalației geotermale de la Universitatea din Oradea.

6.1 Analiza şi specificarea cerinţelor

In procesul de realizare a sistemului de monitorizare şi control, s-a pornit iniţial prin elaborarea documentului de analiză şi specificare a cerinţelor. In cadrul acestuia, au fost sintetizate în detaliu informaţiile despre sistem în ansamblul său: procesul controlat P împreună cu strategia de control C, S = P + C.

6.1.1. Descrierea procesului fizic de controlat P

Din punct de vedere structural, instalaţia geotermală de la Universitatea din Oradea se compune din trei părţi: ***staţia sondei***, ***staţia de pompare*** şi ***punctul termic***. Funcţionarea generală a sistemului implică următoarele [Zma95]:

- apa este extrasă de la staţia sondei, utilizînd o pompă de adîncime care este pusă în funcţiune doar dacă debitul necesar este mai mare decît cel artezian; se utilizează apoi un bazin pentru stocarea apei extrase, atît pe post de acumulator (astfel încît debitul apei de la sondă să nu fie nevoit să urmeze variaţiile în timp ale necesarului de apă în reţea, rezultînd astfel un sistem mult mai stabil) cît şi pentru a separa producţia de reţeaua de distribuţie (adică eventualele disfuncţionalităţi pe termen scurt în furnizarea apei de la sondă să nu afecteze distribuţia)

- ulterior, apa este pompată, prin intermediul staţiei de pompare, către punctul termic, unde nu este utilizată direct, ci prin intermediul unor schimbătoare de căldură. Apa rezultată de pe circuitul de retur se propune a fi utilizată în viitor în mod "cascadă" pentru o seră şi un bazin de înot; de asemenea, pentru a compensa creşterea producţiei şi de a preveni scăderea presiunii în rezervorul subteran de apă geotermală, o parte din apa geotermală utilizată se va reinjecta în sol.

Principalele elemente structurale din care este alcătuit procesul sunt sintetizate în Tabelele 5, 6 şi 7.

Tabelul 5: Elementele componente ale procesului P – ***staţia sondei***

Element	Descriere
P1	Pompă de adîncime
P2	Pompa de ungere a arborelui pompei P1
VSD1	Variator de turaţie pentru controlul pompei P1
CV0	Robinet de reglare pentru controlul debitului de la sondă
M	Motor electric pentru controlul robinetului CV0

Tabelul 6: Elementele componente ale procesului P – ***stația de pompare***

Element	Descriere
P3	Pompă de forță pentru menținerea constantă a presiunii în rețeaua de distribuție
P4	Pompa de forță de rezervă
VSD2	Variator de turație pentru controlul pompei P3 respectiv P4

Tabelul 7: Elementele componente ale procesului P – ***punctul termic***

Element	Descriere
P5 – P7	Pompe de circulație; două dintre acestea sunt utilizate pentru a circula apa în sistemul de încălzire, una este de rezervă
P8 – P9	Pompe de completare: sunt folosite pentru a introduce apă suplimentar în rețea şi de a menține astfel presiunea statică în limite determinate în condițiile existenței unor pierderi
CV2 – CV6	Robinete de reglare
M	Motoare electrice pentru controlul robinetelor CV2-CV6
X2, X4	Schimbătoare de căldură în plăci (perechi)

Schema generală a sistemului împreună cu toate dispozitivele (aparatele de măsură şi control) existente apare în **Anexa 1** (preluată din [Zma95]).

De asemenea, documentul de analiză şi specificare a cerințelor mai conține informații legate de interfața cu utilizatorul. Astfel, pentru a putea monitoriza si controla evoluția sistemului, a existat cerința de a se dezvolta, pe lîngă programul de control propiu-zis şi o interfață cu utilizatorul care să conțină în principal:

- ferestre pentru monitorizarea grafică în timp real a evoluției diferitelor mărimi precizate de utilizator
- o fereastră pentru afişarea alarmelor
- ferestre speciale care vor conține valorile curente ale tuturor constantelor utilizate în cadrul sistemului cu posibilitatea modificării acestora de către operator
- posibilitatea realizării diferitelor calcule şi totalizării (de exemplu, totalizarea cantității totale de apă extrasă pe zi, pe lună, pe an) precum şi a vizualizării acestora în orice moment

- posibilitatea stabilirii diferitelor niveluri de acces pentru operatori, cu parolă.

6.1.2. Descrierea strategiei de control dorite C

Stația sondei are scopul de a extrage apa geotermală de la sondă şi de a o stoca apoi într-un rezervor. Cantitatea de apa care se extrage de la sondă este variabilă, astfel încît nivelul apei din cadrul rezervorului să se mențină constant.

Astfel, la stația sondei, se observă prezența pompei de adîncime P1, utilizată pentru a mări debitul apei la sondă, funcție de necesitățile din sistemul de distribuție. Obiectivul stației sondei este, în conformitate cu strategia de control adoptată, acela de a menține nivelul LT1 din bazin constant. Acest lucru este realizat controlînd fie viteza de operare a pompei P1 (respectiv pornirea/oprirea acesteia), fie prin închiderea/deschiderea robinetului CV0.

Funcția principală a stației de pompare este aceea de a furniza apă caldă gotermală în rețeaua de distribuție, menținînd o presiune constantă în rețea, indiferent de cerințe (presiunea PT1). Acest lucru se realizează controlînd viteza de pompare respectiv pornirea/oprirea pompelor P3 respectiv P4 (de rezervă).

Funcția punctului termic este aceea de a încălzi clădirile Unversității şi de a le furniza apă caldă menajeră. Aceste două procese de încălzire sunt ambele realizate indirect, utilizînduse 4 schimbătoare de căldură în plăci, pentru a furniza energie termică: 2 pentru circuitul de încălzire DH şi 2 pentru circuitul de apă caldă menajeră DHW. În cadrul punctului termic, temperatura TT8 a apei geotermale de la ieşirea din schimbătoarele de căldură precum şi presiunea PT3 a apei calde la ieşirea din schimbătoare sunt mărimi care trebuie menținute la valori constante, asigurînd astfel un randament cît mai bun a instalației, din punct de vedere al energiei furnizate.

Pentru a realiza aceste deziderate, dispozitivele care trebuie controlate sunt pompele de circulație a apei P5-P7 care sunt pornite respectiv oprite în funcție de necesități precum şi robinetele CV2 (pentru controlul temperaturii TT8) şi respectiv CV4 (pentru controlul presiunii PT3).

In etapa următoare, de control algoritmic, este ilustrată modalitatea de realizare efectivă a controlului pe baza strategiei prezentate mai sus.

6.2. Controlul algoritmic

6.2.1. Etapa 1: Realizarea modelului descriptiv M_d

In vederea realizării modelului descriptiv $\mathbf{M_d}$ al sistemului s-a pornit de la modelul descriptiv al procesului $\mathbf{P_d}$, şi care a fost completat ulterior cu strategia de control adoptată. Descrierea modelului descriptiv a fost realizată utilizînd editorul grafic **ACSL/Graphic Modeller** [MGA96a], care permite descrierea la nivel de schemă bloc a tuturor elementelor componente ale acestuia, precum şi a relațiilor existente între ele.

6.2.1.1. Descompunerea sistemului în subsisteme şi definirea modelului descriptiv al procesului P_d

Bazat pe criteriile de descompunere în subsisteme prezentate în paragraful 5.2.1.1., aplicația a fost descompusă în subsisteme pe criterii funcționale, în varianta top-down.

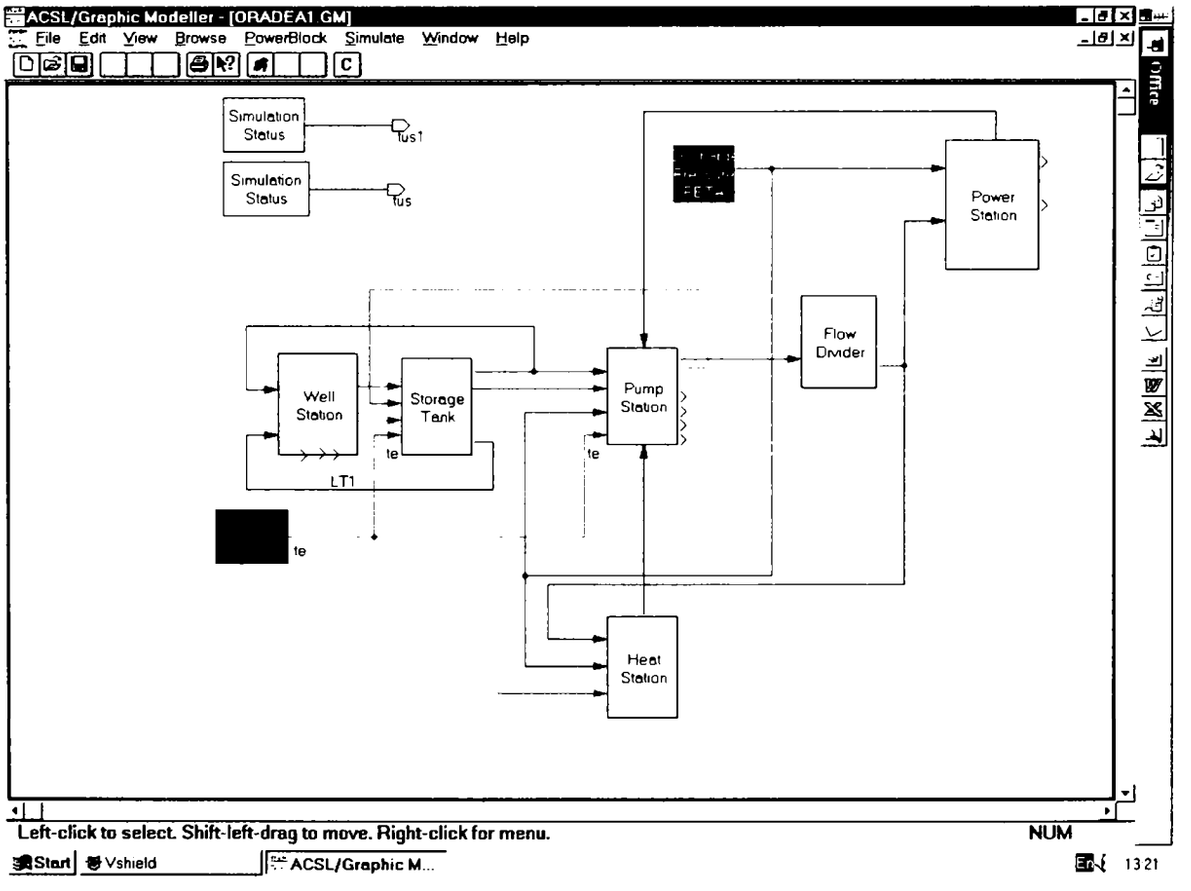


Figura 26: Diagrama schematică a procesului $\mathbf{P_d}$ din Graphic Modeller

Subsistemele procesului fizic de controlat $\mathbf{P_d}$ au fost identificate pe baza descompunerii funcționale şi ele sunt, după cum s-a mai precizat anterior, următoarele: ***stația sondei*** împreună cu bazinul de stocare, ***stația de pompare*** şi ***centrala termică***.

Această împărțire de nivel foarte general este rafinată ulterior în cadrul modelului $\mathbf{P_d}$ pentru fiecare subsistem în parte, pînă cînd se ajunge la nivelul de element de bază (de exemplu, un robinet, un motor, etc.). Acestă rafinare a fost realizată sub forma unei ierarhii: astfel, cu ajutorul editorului grafic **Graphic Modeller** s-a realizat descompunerea în termeni de diagrame bloc utilizînd icoane predefinite sau definite de utilizator în cadrul cărora:

- fiecare bloc identifică un subsistem, şi se caracterizează prin ***numele*** acestuia (ales sugestiv pentru blocul în cauză, de exemplu **We** pentru blocul Well station – stația sondei)
- săgețile cu etichete reprezintă ***mărimile de intrare/ieşire*** din blocul respectiv

Diagrama schematică a procesului $\mathbf{P_d}$ realizată utilizînd acest software este prezentată în Figura 26. Față de descompunerea prezentată mai sus, diagrama mai cuprinde şi blocul aferent legăturii cu centrala electrică (**Flow divider**) şi cel pentru centrala electrică (**Power station**), a căror realizare este în curs de implementare dar nu a fost finalizată pînă în acest moment.

Astfel de diagrame au fost realizate pentru fiecare subsistem în parte; ele vor fi însă prezentate ulterior, după completarea lor şi cu strategia de control adoptată.

6.2.1.2. Definirea strategiei de control C_d

Practic, în procesul de definire a strategiei de control $\mathbf{C_d}$ se porneşte de la cerințele specificate în documentul de analiza şi specificarea cerințelor; în această fază însă definirea controlului a fost realizată în detaliu, specificîndu-se pentru fiecare subsistem în parte mărimea controlată, modalitatea prin care se realizează controlul, diferiți parametrii precum şi diferitele mărimi de intrare/ieşire care sunt asociate controlului [SC97].

In continuare se va prezenta pe scurt o sinteză a strategiei de control adoptate în cazul sistemului de monitorizare şi control a instalației geotemale de la Universitatea din Oradea, defalcată pentru cele trei subsisteme existente: **stația sondei**, **stația de pompare** şi **punctul termic**.

a). Stația sondei

1. Modul de operare

Stația sondei poate fi operată de la distanță în două moduri:

- modul **grup on**: nivelul LT1 din rezervor este controlat alternativ prin intermediul robinetului CV0 sau prin intermediul pompei P1 (automat)
- modul **grup off**: operatorul trebuie să comande robinetul CV0 sau pompa P1 local sau de la distanță, prin intermediul interfeței utilizator

Operarea locală se foloseşte atunci cînd sistemul de control nu funcționează sau în caz de urgență. Operarea locală se realizează utilizînd tabloul de comandă A4 amplasat chiar în interiorul stației sondei.

2. Interfața cu sistemul fizic

2.1. Intrări

Intrări analogice	**Notație folosită**
Transmițător de nivel pentru bazin Scalare: 4-20 mA corespunzător la 0 - 6.0 m înălțime în bazin	LT1
Deschiderea robinetului CV0 Scalare: 4-20 mA corespunzător la 0 -100% deschidere	PT0
Transmițător de presiune pentru presiunea în vîrful sondei Scalare: 4-20 mA corespunzător la 0 - 10.0 bar	PT9
Transmițător de presiune pentru presiunea în puț Scalare: 4-20 mA corespunzător la 0 - 10.0 bar	PT10
Transmițător de curent pentru motorul pompei de adîncime Scalare: 4-20 mA corespunzător la 0 - 100 A	P1_MC
Transmițător de debit pentru debitul apei la sondă Scalare: 4-20 mA corespunzător la 0 - 50.0 l/s	FT1
Frecvența de operare a pompei P1 Scalare: 4-20 mA corespunzător la 0 - 100 Hz	P1_HZ
Transmițător de vibrații pentru pompa P1 Scalare: 4-20 mA corespunzător la 0 - 5.0 mm/s^2	VT1
Transmițător de temperatură pentru temperatura apei la sondă Scalare: 4-20 mA corespunzător la 0 - 120 °C	TT1

Intrări digitale	Notația folosită
Switch indicînd deschiderea completă a robinetului CV0	CV0_LO
Switch indicînd închiderea completă a robinetului CV0	CV0_LC
Switch indicînd operarea robinetului CV0 în direcția de deschidere	CV0_TL1
Switch indicînd operarea robinetului CV0 în direcția de închidere	CV0_TL2
Defecțiune la valva CV0	CV0_CA
Switch auxiliar alimentare cu curent pompa P1	P1_ON
Switch pentru protecție supraîncărcare/scurt circuit pompa P1	P1_OL
Switch termic pompa P1	P1_TD
Switch indicînd VSD1 în stare de funcționare	VSD1_RD
Switch indicînd VSD1 defect	P1_FA
Switch auxiliar alimentare cu curent pompa P2	P2_ONST
Switch termic pompa P2	P2_TD
Switch pentru protecție supraîncărcare/scurt circuit pompa P2	P2_OL
Temperatură prea mare în stația sondei	TS1
Umiditate prea mare în stația sondei	AS1
Switch pentru selecție comandă de la distanță/locală stația sondei	RL2
Switch indicînd oprire locală de urgență de la panoul stației sondei	ES2
Counter pentru debitul apei Scalare: 1 puls corespunzător la 100 l apă	FT1_CN
Switch indicînd direcția de curgere a apei/eroare	FT1_ER
Switch indicînd presiunea prea mică a apei de ungere pentru pompa P2	PS1

2.2. Ieşiri

Ieşiri digitale	Notația folosită
Start P1, puls pentru 1.0 s	P1_RNRN
Stop P1, puls pentru 1.0 s	P1_STST
VSD1 start/stop	VSD1
Deschide robinetul CV0	CV0_OP
Închide robinetul CV0	CV0_CL
Switch pentru resetarea transmițătorului de vibrații VT1	VT1_RZ

Ieşiri analogice	Notația folosită
Viteza de rotație pentru VSD1, VSD1_SP Scalare: 4-20 mA corespunzător la 0-100%	VSDP1_SP

3. Controlul algoritmic

Dispozitive controlate	Notația folosită
Pompa P1 asociată cu variatorul de turație VSD1	P1 + VSD1
Pompa P2 acționată on/off	P2
Robinetul CV0 asociat cu un motor electric M on/off pentru cele două direcții deschidere/închidere	CV0

Pompa P1 este controlată cu un variator de turație VSD1. Deschiderea valvei CV0 este controlată cu ajutorul unui motor (ON/OFF în două direcții) cu timpul de deschidere/închidere egal cu 40s. În cazul în care acest timp nu este atins, se va semnala alarmă şi se va trece sistemul pe comandă manuală (grup off). Pompa P2 este controlată ON/OFF.

Două regulatoare se folosesc în acest scop: regulatorul RG1 care controlează nivelul LT1 prin deschiderea/închiderea valvei CV0 şi regulatorul RG2 care controlează nivelul LT1 prin pornirea/oprirea pompei P1. Cele două regulatoare nu pot funcționa automat decît alternativ, în modul grup on. În modul grup off, regulatoarele sunt trecute în regim manual, cu o ieşire fixată, şi în acest caz operatorul trebuie să exercite controlul manual asupra lui CV0 sau P1, local sau de la distanță, prin intermediul interfeței utilizator.

Operarea pompei de adîncime P1:

- se contorizează orele de funcționare ale pompei P1 folosind contorul c_{w1}.
- se contorizează numărul de porniri/oră ale pompei P1 folosind contorul c_{w2}; dacă acesta depăşeşte x_{w2} ori/oră, stația este pusă în modul grup off, blocîndu-se alte porniri automate ale lui P1
- dacă LT1 indică rezervor aproape gol (alarma LL) şi RG2 are ieşire 100% pentru mai mult de t_{w4} minute, se pune regulatorul în regim manual cu ieşirea 0% ,opreşte P1 şi se trimite o alarmă specială către utilizator; o listă completă a alarmelor din cadrul sistemului este prezentată în **Anexa 2**.

- dacă LT1 indică rezervor aproape plin (alarma HH) şi RG2 are ieşire 0% pentru mai mult de t_{w3} minute, se pune regulatorul în regim manual cu ieşirea 0%, opreşte P1 şi se trimite o alarmă specială către utilizator
- dacă presiunea furnizată în rețea PT1 este prea mare (alarma HH) pentru mai mult de t_{p4} secunde, se opreşte pompa P1

Operarea valvei CV0:

- dacă CV0 dă eroare de limită de închidere pentru mai mult de t_{L1} secunde, se va opri operarea acesteia, şi se va trece regulatorul RG1 în regim manual, cu ieşire fixată
- analog, dacă CV0 dă eroare de limită de deschidere pentru mai mult de t_{L2} secunde, se va opri operarea acesteia, şi se va trece regulatorul RG1 în regim manual, cu ieşire fixată
- dacă CV0 dă eroare de mişcare pentru mai mult de t_{L3} secunde, se va opri operarea acesteia, şi se va trece regulatorul RG1 în regim manual, cu ieşire fixată

Regulatoarele prezentate în Tabelul 8 sunt utilizate pentru realizarea controlului în maniera descrisă mai sus :

Tabelul 8: Regulatoarele utilizate la stația sondei

Regulator	**Mărimea controlată**	**Elementul final de control**
RG1	nivelul apei în rezervor LT1	robinetul de reglare CV0
RG2	nivelul apei în rezervor LT1	variatorul de turație VSD1
RG11	presiunea în rețeaua de distribuție PT1	robinetul de reglare CV0
RG12	presiunea în rețeaua de distribuție PT1	variatorul de turație VSD1

Regulatoarele RG1 şi RG2 nu pot fi amîndouă simultan în regim automat:

- RG1 trebuie să fie pe auto la regim artezian, iar atunci RG2 trebuie să fie pe manual 0%
- RG2 trebuie să fie pe auto cînd P1 este operată, iar RG1 pe manual 100%

In mod analog se comportă şi perechea de regulatoare RG11 respectiv RG12:

- RG11 trebuie să fie pe auto la regim artesian, iar atunci RG12 trebuie să fie pe manual 0%

- RG12 trebuie să fie pe auto cînd P1 este operată, iar RG11 pe manual 100%

Doar una dintre cele două perechi de regulatoare este utilizată la un anumit moment dat, selecția realizîndu-se de către operator.

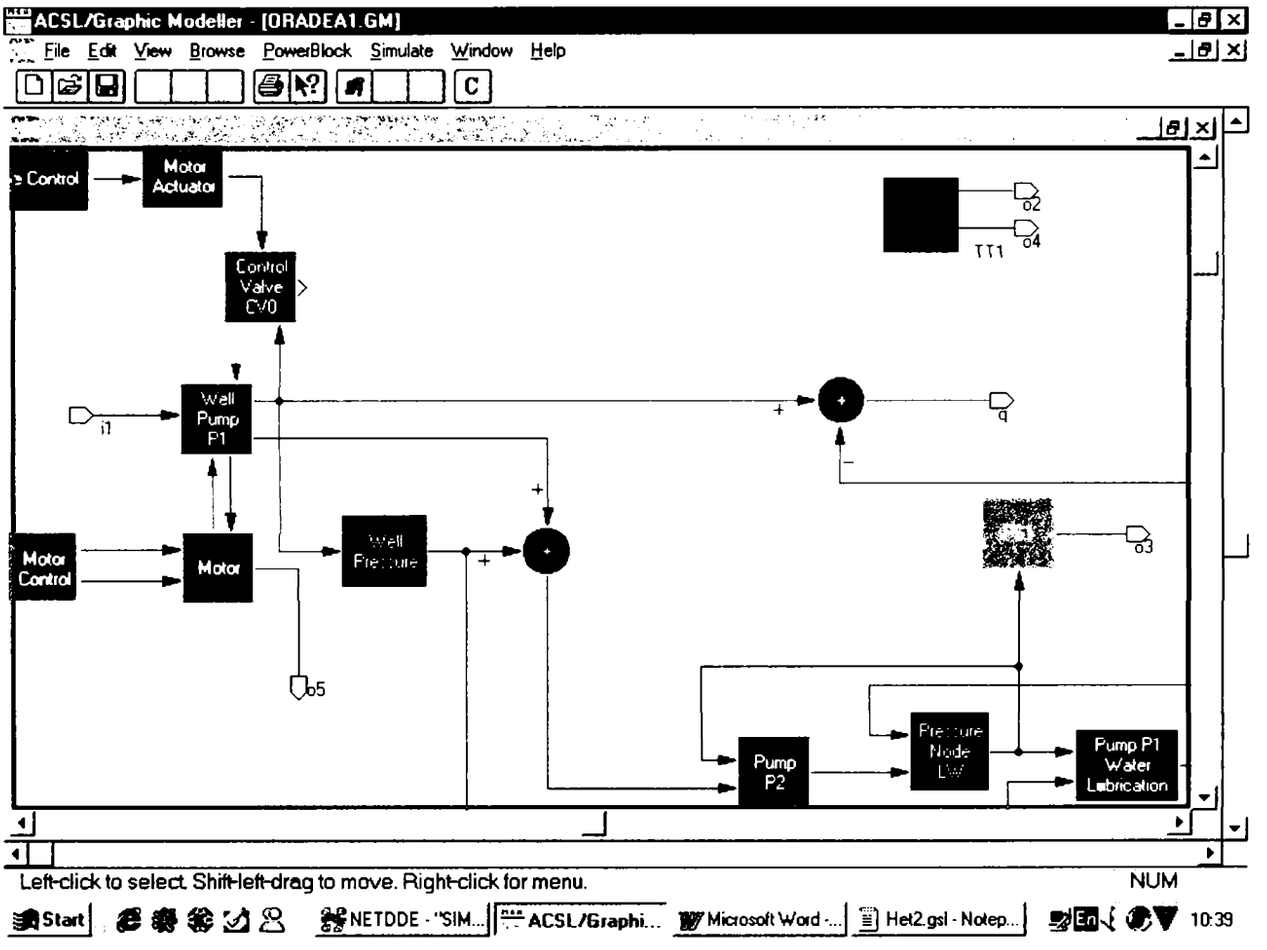


Figura 27: Modelul descriptiv pentru stația sondei

4. Interfața cu utilizatorul

Fereastra corespunzătoare stației sondei conține afişarea următoarelor elemente:

- un desen schematic al stației sondei
- afişarea mărimilor de interes: TT1, PT9, FT1, VT1, PT10, LT1
- afişarea modului de operare grup on/off precum şi posibilitatea de schimbare a acestuia de operator

- afişarea regulatoarelor RG1 şi RG2, cu informaţii suplimentare despre acestea: set-point-ul SP, variabila de proces PV şi ieşirea OP ale acestora, posibilitatea modificării de către operator a SP în regim automat şi a OP în regim manual
- afişarea de informaţii despre starea pompelor (pornit/oprit, defectă) cu posibilitatea pornirii şi respectiv opririi de către operator a acestora în regim manual

Conform metodologiei prezentate în capitolul 5, definitivarea modelului $\mathbf{M_d}$ presupune combinarea modelului obţinut pentru $\mathbf{P_d}$ cu strategia de control adoptată $\mathbf{C_d}$ într-un model unic, şi care va repezenta modelul descriptiv al întregului sistem de control în timp real, $\mathbf{M_d}$. Astfel, pornind de la modelul $\mathbf{P_d}$ descris în **ACSL/Graphic Modeller**, acesta a fost completat cu blocuri de control aferente tuturor elementelor controlate, conform celor prezentate mai sus.

In final, modelul descriptiv rezultat $\mathbf{M_d}$ se prezintă sub forma unei ierarhii de blocuri şi sub-blocuri, care au ca punct de plecare diagrama din Figura 26. Figura 27 arată modelul descriptiv $\mathbf{M_d}$ rezultat pentru staţia sondei.

b). Staţia de pompare

1. Moduri de operare

Staţia de pompare poate fi operată de la distanţă în două moduri:

- modul **grup on**: presiunea PT1 este controlată prin intermediul pompei P3 respectiv P4 (automat)
- modul **grup off**: operatorul trebuie să comande pompa P3 sau P4 local sau de la distanţă, prin intermediul interfeţei utilizator

Operarea locală se foloseşte atunci cînd sistemul de control nu funcţionează sau în caz de urgenţă. Operarea locală se realizează utilizînd tabloul de comandă A3 amplasat chiar în înteriorul staţiei de pompare.

2.1. Intrări

Intrări analogice	**Notație folosită**
Transmițător de presiune pentru presiunea în sistem Scalare: 4-20 mA corespunzător la 0 - 10.0 bar	PT1
Transmițător de temperatură pentru temperatura mediului exterior (media valorilor pentru ultimele 10 minute)	TT14
Transmițător pentru viteza vîntului (media valorilor pentru ultimele 10 minute)	WT1

Intrări digitale	**Notația folosită**
Switch auxiliar alimentare cu curent pompa P3 /P4	P3_ON, P4_ON
Switch pentru protecție supraîncărcare/scurt circuit pompa P3/P4	P3_OL, P4_OL
Switch termic pompa P3/P4	P3_TD, P4_TD
Switch indicînd VSD2 în stare de funcționare	VSD2_RD
Switch indicînd VSD2 defect	P3_FA
Temperatură prea mare în stația sondei	TS2
Umiditate prea mare în stația sondei	AS2
Switch pentru selecție comandă de la distanță/locală stația de pompare	RL1
Switch indicînd oprire locală de urgența de la panoul stației de pompare	ES1
Switch indicînd presiune prea mică de absorbție	PS2
Switch indicînd presiune de refulare prea mare	PS3

2.2. Ieşiri

Ieşiri analogice	**Notația folosită**
Viteza de rotație pentru VSD2, VSD2_SP Scalare: 4-20 mA corespunzător la 0-100%	VSDP2_SP

Ieşiri digitale	Notația folosită
Start P3/P4, puls pentru 1.0 s	P3_RNRN, P4_RNRN
Stop P3/P4, puls pentru 1.0 s	P3_STST, P4_STST
VSD2 start/stop	VSD2

3. Controlul algoritmic

Dispozitive controlate	Notația folosită
Pompa P3/P4 asociată cu variatorul de turație VSD2	P3 + VSD2, P4 + VSD2
Pompa P3/P4 acționată on/off	P3, P4

Pompele P3/P4 sunt controlate de un variator de turație VSD2. Regulatorul RG3 se foloseşte în acest scop; acesta controlează presiunea în rețea PT1 prin variația turației motorului pompelor P3/P4, utilizînd variatorul de turație VSD2.

Operarea pompelor de circulație P3 şi P4:

- se contorizează orele de funcționare ale pompelor P3/P4 folosind contoarele c_{p1} şi c_{p2}.
- se contorizează numărul de porniri/oră ale pompelor P3/P4 folosind contorul c_{p3} şi c_{p4}; dacă acesta depăşeşte x_{p1} ori/oră, stația este pusă în modul grup off, blocîndu-se alte porniri automate ale lui P3 respectiv P4
- dacă semnalul de la PT1 este intrerupt mai mult decît t_{p1} secunde, se trece stația de pompare în modul group off, iar regulatorul RG3 în mod manual cu ieşirea fixată
- dacă presiunea furnizată în rețea PT1 este prea mare (alarma HH) pentru mai mult de t_{p4} secunde, se trece stația de pompare în modul group off, iar regulatorul RG3 în mod manual cu ieşirea fixată
- dacă transmițătorul de nivel LT1 indică alarma L (nivel prea mic al apei din rezervor) pentru mai mult de t_{p5} minute, se opresc pompele P3 respectiv P4, şi se pune regulatorul RG3 în mod manual cu ieşirea fixată

Regulatorul din Tabelul 9 este folosit pentru realizarea controlului descris mai sus:

Tabelul 9: Regulatoarele utilizate în stația de pompare

Regulator	Mărimea controlată	Elementul final de control
RG3	presiunea furnizată în rețea PT1	variatorul de turație VSD2

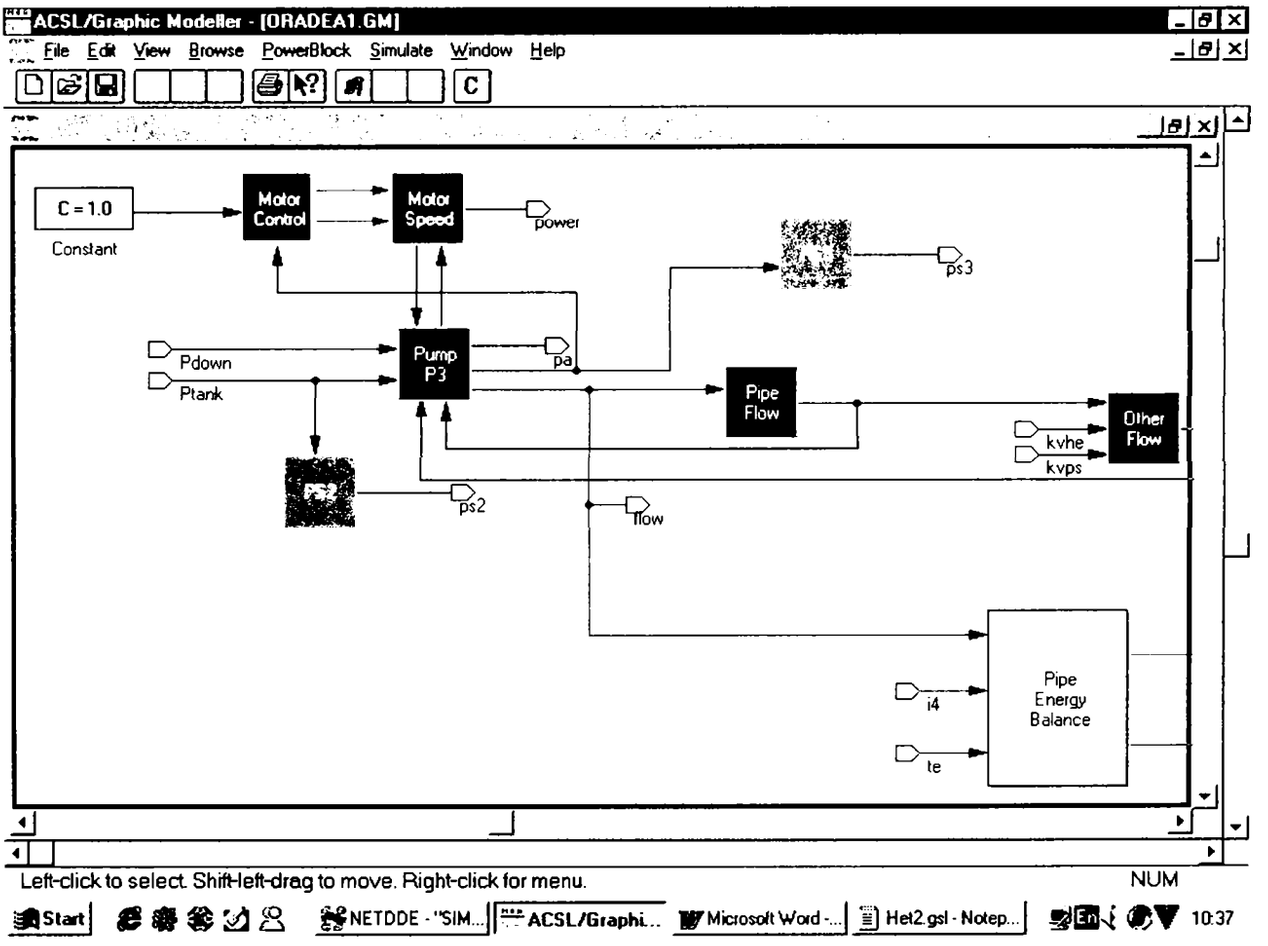


Figura 28: Modelul descriptiv pentru stația de pompare

4. Interfața cu utilizatorul

Fereastra corespunzătoare stației de pompare conține afișarea următoarelor elemente:

- un desen schematic al stației de pompare
- afişarea mărimilor de interes: PT1

- afişarea modului de operare grup on/off precum şi posibilitatea de schimbare a acestuia de operator
- afişarea regulatorului RG3 cu informaţii suplimentare despre acesta: set-point-ul SP, variabila de proces PV şi ieşirea OP ale acestora, posibilitatea modificării de către operator a SP în regim automat şi a OP în regim manual
- afişarea de informaţii despre starea pompelor (pornit/oprit, defectă) cu posibilitatea pornirii şi respectiv opririi de către operator a acestora în regim manual

Modelul descriptiv rezultat pentru staţia de pompare, în implemetarea cu ajutorul editorului grafic ACSL/Graphic Modeller, este prezentat în Figura 28.

c). Punctul termic

1. Moduri de operare

Punctul termic poate fi operat de la distanţă în două moduri:

- modul **grup on**: pompele de circulaţie şi de completare sunt pornite/oprite automat, funcţie de validitatea condiţiilor de pornire/oprire a acestora
- modul **grup off**: operatorul trebuie să comande pompele din punctul termic prin intermediul interfeţei utilizator; sunt valabile în acest caz aceleaşi condiţii de pornire/oprire a pompelor ca şi în modul grup on

In modul de operare local, acţionarea pompelor poate fi realizată de la panoul de comandă A2, amplasat în interiorul punctului termic. Operarea locală se utilizează atunci cînd sistemul de control se defectează sau în caz de urgenţă.

2. Interfaţa cu sistemul fizic

2.1. Intrări

Intrări analogice	**Notaţie folosită**
Transmiţătoare de presiune pentru presiunea apei în diferite puncte de pe circuitul din punctul termic Scalare: 4-20 mA corespunzător la 0 - 10.0 bar	PT3 - PT8

Transmițătoare de temperatură pentru temperatura apei în diferite puncte de pe circuitul din punctul termic Scalare: 4-20 mA corespunzător la 0 - 120 °C	TT4 -TT13
Transmițătoare de debit pentru debitul apei în diferite puncte de pe circuitul din punctual termic Scalare: 4-20 mA corespunzător la 0 - 50.0 l/s	FT3-FT6

Intrări digitale	**Notația folosită**
Switch-uri auxiliare alimentare cu curent pompele P5-P9	P5_ON .... P9_ON
Switch-uri pentru protecție supraîncărcare/scurt circuit a pompelor P5-P9	P5_OL.... P9_OL
Switch-uri termice pentru pompele P5-P9	P5_TD.... P9_TD
Temperatură prea mare în punctul termic	TS3
Umiditate prea mare în punctul termic	AS3
Switch pentru selecție comandă de la distanță/locală punctul termic	RL3
Switch indicînd oprire locală de urgența de la panoul punctului termic	ES3
Switch indicînd presiune prea mică în amortizoarele pneumatice	PS4

2.2. Ieşiri

Ieşiri digitale	**Notația folosită**
Start P5 – P9, puls pentru 1.0 s	P5_RNRN....P9_RNRN
Stop P5 – P9 puls pentru 1.0 s	P5_STST.... P9_STST

3. Controlul algoritmic

Dispozitive controlate	**Notația folosită**
Pompele P5 – P9 acționate on/off	P5 – P9
Robineții CV2-CV6 asociați cu cîte un motor electric M on/off pentru cele două direcții respectiv deschidere/închidere	CV2 – CV6

Operarea pompelor de circulație P5 - P7:

- se contorizează orele de funcționare ale pompelor P5 – P7 folosind contoarele c_{h5} - c_{h7}
- se contorizează numărul de porniri/oră ale pompelor P5 – P7 folosind contoarele c_{h1} - c_{h3}
- în modul grup on, controller-ul va acționa două pompe. La pornirea pompelor va porni cu aceea care are cel mai mic timp de lucru, va aştepta t_{h5} secunde şi apoi va porni aceea pompă care are al doilea timp de utilizare. Controller-ul nu poate opera simultan mai mult de două pompe. Astfel, dacă în modul grup on una dintre pompe este oprită de operator, atunci se va porni pompa de rezervă instantaneu.
- dacă:
  - PT4 - presiunea furnizată în rețea este prea mare (alarmă HH), pentru mai mult de t_{h6} secunde
  - PT5 - presiunea retur este prea scăzută (alarmă LL), pentru mai mult de t_{h7} secunde
  - PT6 - presiunea evacuare este prea mare (alarmă HH), pentru mai mult de t_{h8} secunde
  - (PT6 - PT3) - presiunea diferențială pe schimbătoarele de căldură este prea mare (mai mare decît o valoare dată x_{h2}), pentru mai mult de t_{h9} secunde

  atunci controller-ul opreşte pompa de circulație cu cel mai mare timp de funcționare şi pune punctul termic în modul grup off. Dacă alarma este încă validă după t_{h10} sec, atunci se opreşte şi a doua pompă şi se semnalează alarmă către operator.

Operarea pompelor de compensare P8 şi P9

- se contorizează orele de funcționare ale pompelor P8 – P9 folosind contoarele c_{h8} - c_{h9}
- contoarele c_{h10} şi c_{h11} numără de cîte ori au fost pornite pompele P8 şi P9, independent de modul de operare al stației. Dacă punctul termic este în modul grup on şi numărul de porniri ale uneia dintre pompe depăşeşte x_{h7} ori / oră, atunci controller-ul opreşte acea pompă şi porneşte pompa de rezervă. Schimbarea pompei active este înregistrată. Dacă acest eveniment se repetă în t_{h1} minute de la schimbare, atunci controller-ul nu mai porneşte pompa şi raportează o problemă în sistem, interpretînd situația ca o pierdere excesivă în rețea şi punînd stația în modul grup off

- în modul grup on controller-ul nu poate acționa la un anumit moment dat decât o singură pompă. Pompa ce va fi acționată este selectată cu un comutator de către operator. De exemplu, dacă pompa P8 este selectată, atunci controller-ul o porneşte atunci cînd presiunea de retur PT5 coboară sub x_{h4} barg. Atunci când PT5 ajunge la x_{h5} barg, pompa este oprită. Situația se prezintă în mod similar pentru cazul în care P9 este activă.
- în modul grup off operatorul este cel care trebuie să pornească pompele de compensare. Acesta poate face aceasta doar dacă presiunea de retur PT5 este sub limita maximă, x_{h5}. Pompele se opresc automat la atingerea limitei superioare.
- dacă controller-ul trebuie să acționeze continuu pompa de compensare pentru mai mult de t_{h2} ore odată, atunci va opri pompa şi va raporta o problemă în sistemul de compensare (alarmă), interpretînd situația ca fiind o pierdere excesivă în rețea şi punînd punctul termic în modul grup off.

Următoarele regulatoare prezentate în Tabelul 10 sunt folosite pentru realizarea controlului conform celor precizate mai sus:

Tabelul 10: Regulatoare utilizate în punctul termic

Regulator	**Mărimea controlată**	**Elementul final de control**
RG5	temperatura apei introduse în rețeaua de încălzire TT8	robinetul de reglare CV2
RG6	presiunea la intrarea în schimbătoarele de căldură PT3	robinetul de reglare CV4
RG7	temperatura apei introduse în rețeaua de apă caldă menajeră TT12	robinetul de reglare CV3
RG8	presiunea apei în schimbătorul de căldură PT7	robinetul de reglare CV5

4. Interfața cu utilizatorul

Fereastra corespunzătoare punctului termic conține afişarea următoarelor elemente:

- un desen schematic al punctului termic
- afişarea mărimilor de interes: FT3 – FT6, PT3 – PT8, TT4 – TT13

- afişarea modului de operare grup on/off precum şi posibilitatea de schimbare a acestuia de operator
- afişarea regulatoarelor RG5 – RG8 împreună cu informaţii suplimentare despre acestea: set-point-ul SP, variabila de process controlată PV precum şi ieşirea OP ale acestora, cu posibilitatea modificării de către operator a SP în regim automat şi a OP în regim manual
- afişarea de informaţii despre starea pompelor (pornit/oprit, defectă) cu posibilitatea pornirii şi respectiv opririi de către operator a acestora în regim manual

Modelul descriptiv rezultat pentru punctul termic, în implemetarea cu ajutorul editorului grafic ACSL/Graphic Modeller, este prezentat în Figura 29.

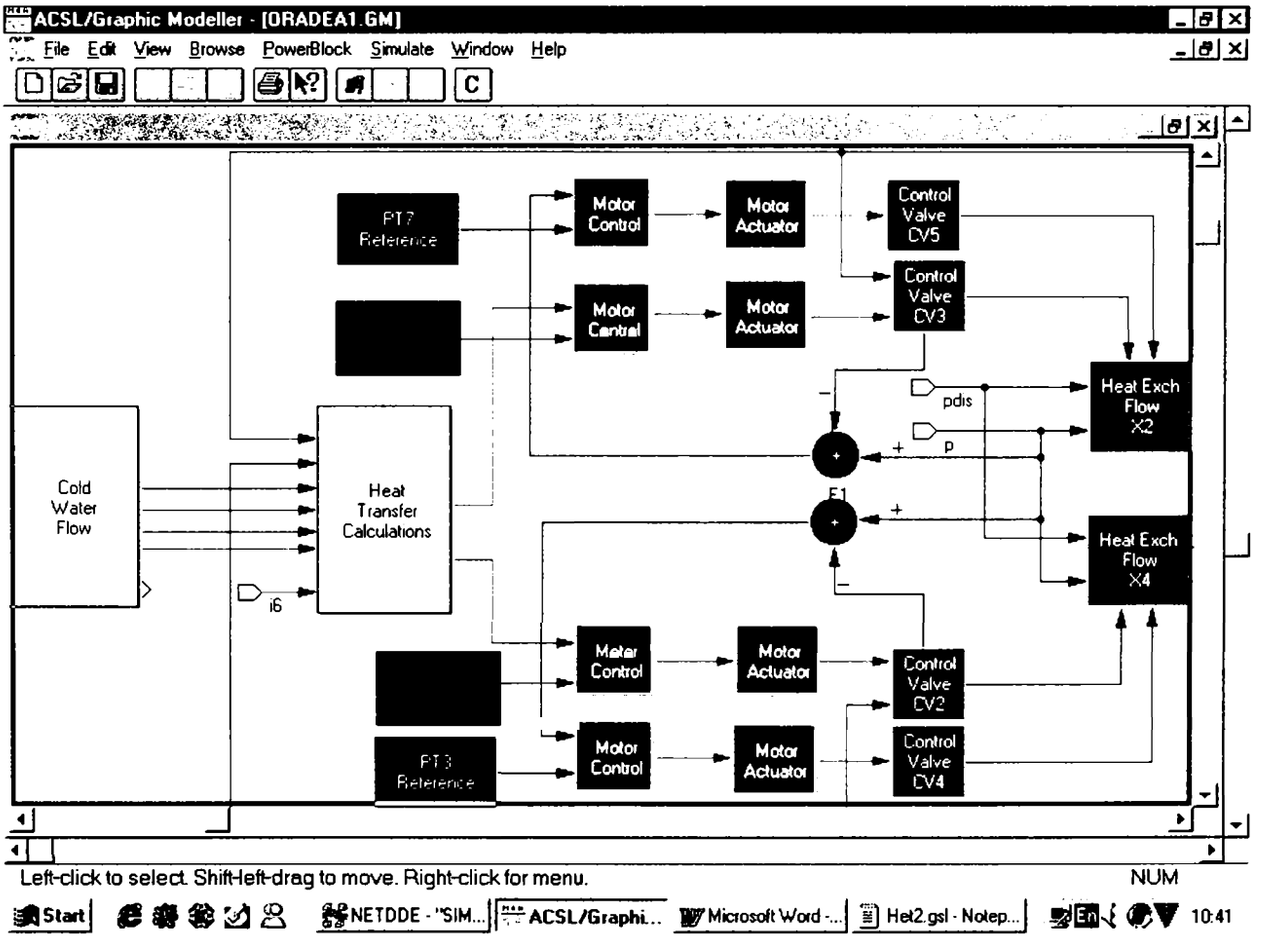


Figura 29: Modelul descriptiv pentru punctul termic

Anexa 3 prezintă în completarea descrierii strategiei de control de mai sus, o listă completă a tuturor constantelor utilizate în cadrul acesteia (counter-e, constante de timp, constante utilizate pentru memorarea valorilor minime respective maxime premise ale anumitor mărimi, diferite valori calculate, etc.). Utilizarea acestor constante a fost deja prezentată; în sistemul real, operatorul (dacă beneficiază de nivelul de acces corespunzător), are posibilitatea schimbării interactive a acestora, prin intermediul interfeței utilizator.

In concluzie, modelele descriptive rezultate pentru stația sondei, stația de pompare şi punctual termic alcătuiesc împreună modelul descriptiv complet al sistemului, $\mathbf{M_d} = \mathbf{P_d} + \mathbf{C_d}$.

6.2.2. Etapa 2: Realizarea modelului prescriptiv formal M_p şi verificarea acestuia prin simulare

6.2.2.1. Realizarea modelului prescriptiv M_p şi simularea acestuia

In procesul de generare al modelului prescriptiv formal **M_p**, s-a pornit de la modelul **M_d** dezvoltat în etapa anterioară. Procesul de creare a modelului **M_p** a cuprins două etape:

- ***rafinarea*** modelul descriptiv la nivel de bloc **M_d** pînă la faza de ecuații matematice concrete care descriu o anumită porțiune specifică din sistem; practic, fiecare element al sistemului a fost descris prin ecuații matematice: acest lucru se referă atît în cazul elementelor fizice cît şi în cazul blocurilor de control aferente acestora
- ***programarea*** acestor ecuații utilizînd limbajul de simulare **ACSL** în scopul simulării ulterioare a acestora; practic, pentru aceasta s-a utilizat opțiunea **Graphic Modeller** a limbajului **ACSL** [MGA96b][MGA95], care furnizează un front-end uşor de utilizat pentru a genera simulări **ACSL**

In continuare se va prezenta pentru exemplificare modalitatea de generare a ecuațiilor matematice pentru cele mai reprezentative elemente din cadrul sistemului.

a). Modelarea robinetului de reglare CV0 şi a controlului acestuia

Ecuația prin care se modelează robinetul de reglare CV0 este dată de următoarea formulă [IEC95][CVH98]:

$$k = k_{\max}\left(a_1 u^3 + a_2 u^2 + a_3 u\right) \qquad \text{(r6.1)}$$

unde: a_1, a_2, şi a_3 sunt constante specifice robinetului (date de producător)
u este variabila prin care se modelează mărimea de comandă primită de la regulator pentru elementul de execuție
k reprezintă deschiderea robinetului (între limitele 0 şi k_{max})

Căderea de presiune se obține din ecuația generală cunoscută:

$$Q = k^2 \Delta p \qquad \text{(r6.2)}$$

Astfel, pentru cazul particular al robinetului CV0, funcție de deschiderea obținută k, precum şi de debitul apei care vine de la sondă q, se poate calcula

ulterior căderea de presiune a apei după robinetul de reglare, corespunzătoare variabilei PT9:

$$pressure = \max(0.0, (q/(k+0.00001))^2) \qquad \text{(r6.3)}$$

Termenul 0.00001 din ecuația de mai sus este introdus doar din considerente matematice, şi anume pentru a nu avea împărțire cu zero; în formula generală acesta nu intervine.

Debitul apei de la sondă q este calculat anterior în cadrul secvenței care modelează pompa P1, unde acesta este determinat în funcție de caracteristica de debit a robinetului k , mărime calculată în blocul CV0. Practic, aceste două mărimi se intercondiționează într-o buclă ele luând diverse valori bine determinate în timp, dacă valorile inițiale furnizate ACSL-ului sunt corecte.

Valoarea calculată pentru presiunea de mai sus va putea fi afişată şi testată ulterior în cadrul modelului, ca şi cînd ea ar proveni de la un senzor de presiune (PT9 în sistemul real). Blocul referitor la modelarea robinetului CV0, aşa cum apare în cadrul editorului grafic ACSL/Graphic Modeller este ilustrat în Figura 30.

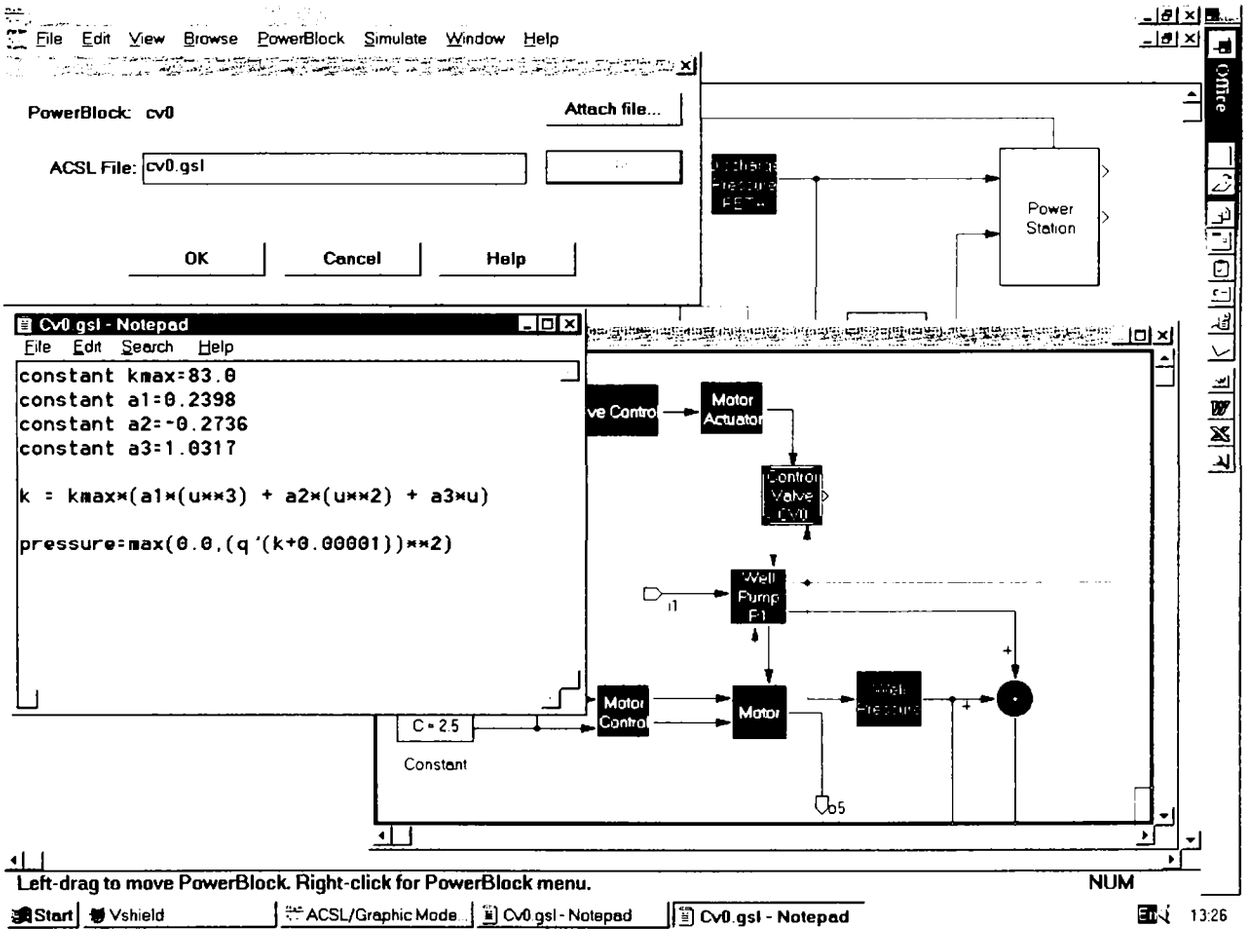


Figura 30: Blocul pentru controlul robinetului CV0

In cadrul fişierului .csl , care prezintă o descriere textuală în limbajul de simulare ACSL a imaginii de mai sus, blocul pentru controlul robinetului CV0 se prezintă sub următoarea formă:

```
ACSL Block: cv0
! Last Change: 02:44:03 12/06/97
! Description:
! Input Ports:
!    Port Name: i1
!    Connected Variable: q
!    Port Name: i3
!    Connected Variable: Weactuator1acth
! Output Ports:
!    Port Name: o1
!    Connected Variable: k
!    Port Name: pressure
!    Connected Variable: pressure
!-------------------------------------------------------------------------
--
CONSTANT Wecv0a1=0.2398
CONSTANT Wecv0a2=-0.2736
CONSTANT Wecv0a3=1.0317
CONSTANT Wecv0kmax=83.0

Wecv0k  = Wecv0kmax *(Wecv0a1 *(Weactuator1acth **3) +
          Wecv0a2 *(Weactuator1acth **2) + Wecv0a3 *Weactuator1acth )
Wecv0pressure =max (0.0,(WeP1q /(Wecv0k +0.00001))**2)
```

Se observă că numele blocului este **cv0**, el făcînd parte din blocul **We** (staţia sondei – Well station) şi în consecinţă toate numele variabilelor utilizate în cadrul blocului au prefixul Wecv0. Această notaţie permite localizarea rapidă în cadrul blocurilor şi sub-blocurilor, pe linie ierarhică, a tuturor variabilelor utilizate. De exemplu, se observă faptul că variabila WeP1q provine din cadrul sub-blocului P1 din cadrul blocului We, după cum am precizat anterior.

Valoarea variabilei *u* din cadrul ecuaţiei (r6.1) este dată de ieşirea regulatorului RG1 care, utilizînd un algoritm de control PID, acţionează actuatorul pentru închiderea respectiv deschiderea robinetului CV0 funcţie de nivelul apei din rezervor (Tabelul 8).

Implementarea controlului PID este realizată conform binecunoscutei ecuaţii generale [PH96]:

$$u = K_p + \frac{1}{T_i}\int_0^t e\,dt + T_d\frac{de}{dt} \qquad \text{(r6.4)}$$

în cadrul căreia: K_p = constantă de proporţionalitate
T_i = constantă de integrare
T_d = constantă de derivare

Aceste constante sunt cunoscute, însă ele pot fi modificate interactiv de către operator pentru a obține o mai bună performanță în procesul de reglare. In cadrul ecuației (r6.4), variabila *e* reprezintă eroarea calculată ca şi diferența între valoarea dorită (setpoint-ul SP) şi valoarea actuală a mărimii controlate de regulator.

In cazul concret al regulatorului RG1, mărimea controlată este nivelul LT1 din rezervor (variabila Tatl1h în model) iar setpoint-ul îl reprezintă o constantă (variabila WeConstanto1); eroarea calculată este notată în acest caz cu *SummerE*, şi deci ecuația (r6.4) devine:

$$u = K_p + \frac{1}{T_i}\int_0^t SummerE\,dt + T_d\frac{dSummerE}{dt} \qquad \text{(r6.5)}$$

Sub-blocul Summer arată modalitatea de calcul a variabilei SummerE:

```
! Summation Block: Summer
! Last Change: 10:43:08 09/18/97
! Description:
! Input Ports:
!    Port Name: i1
!    Connected Variable: Tatl1h
!    Port Name: i2
!    Connected Variable: WeConstanto1
! Output Ports:
!    Port Name: E
!-------------------------------------------------------------------------
--
WepidlSummerE = +WeConstanto1-Tatl1h
```

b). Modelarea motorului M2 cu turație variabilă asociat pompei de adîncime P1

Motorul M2 reprezintă motorul de acționare a pompei de adâncime P1. Variabilele de intrare ale blocului care modelează motorul M2 sunt tensiunea *u* şi frecvența de alimentare *f* (care sunt furnizate într-un alt bloc, prin intermediul variatorului de turație VSD2) precum şi cuplul rezistent al pompei *tpump*, care vine de la blocul prin care se modelează pompa de adâncime P1 [Pop86]. Ca şi la cealaltă ramură de reglare, a robinetului CV0, avem două blocuri pentru care ieşirea unuia este intrare pentru celălalt şi invers. Şi în acest caz valorile vor fi generate de ACSL în buclă, pornind de la valorile (condițiile) inițiale. Mărimile de ieşire ale blocului M2 sunt turația motorului *n* (şi deci şi a pompei de care este cuplat rigid) şi puterea consumată a motorului *power*.

Ecuațiile care modelează comportarea motorului M2 cu turație variabilă sunt descrise în continuare. Astfel, pe baza frecvenței de alimentare *f* se calculează viteza unghiulară ω_1:

$$\omega_1 = 2 * \pi * f \qquad \text{(r6.6)}$$

rezultînd caracteristica de modificare de turație *s*:

$$s = (\omega_1 - \omega) \,/\, (\omega_1 + 0.00001) \qquad \text{(r6.7)}$$

unde ω reprezintă viteza unghiulară nominală. In continuare, pe baza constantelor cunoscute ale motorului, se calculează mărimile intermediare *r2s* respectiv x_1 şi x_2, cu ajutorul cărora se va calcula ulterior momentul motor *Tem*:

$$r2s = (r21/(s + 0.00001)) + r22 + r23 * s\,, \qquad \text{(r6.8)}$$

r21, r22, r23 constante

$$x_1 = 2 * \pi * f * l1$$
$$x_2 = 2 * \pi * f * l2 \qquad \text{(r6.9)}$$

l1, l2 constante

Se calculează apoi momentul motor *Tem* prin înlocuirea valorilor calculate în relațiile <15> - <18>, conform formulei:

$$Tem = \left(\frac{VSD2v}{\omega_1}\right)^2 * \frac{m1 * r2s}{(r1 + r2s)^2 + (x1 + x2)^2} \qquad \text{(r6.10)}$$

Ecuațiile prin care se determină mărimile calculate sunt ecuațiile obișnuite pentru un motor electric asincron. Pentru acesta, viteza unghiulară momentană (ω) este definită ca integrală a accelerației unghiulare ($d\omega$) pentru condiția inițială dată (ω_0, care în acest caz este zero). Accelerația unghiulară se obține din valorile momentane ale cuplului motor *Tem* şi al cuplului rezistent *Tpump*, a căror diferență este împărțită la modulul de inerție polar *Jm*. Cuplul rezistent este o variabilă de intrare, iar cuplul motor se calculează în funcție de tensiunea aplicată (care este tot o mărime de intrare) şi de viteza unghiulară *ω1*, calculată din frecvența primită de la VSD2. În final se obțin din ecuații simple turația şi puterea consumată a motorului

In consecință, rezultă:

$$\frac{d\omega}{dt} = \frac{Tem - tpump}{jm} \qquad \text{(r6.11)}$$

, *jm* moment de inerție polar

şi deci prin integrare, se poate calcula ω. Cu ω astfel calculat, se poate calcula turația motorului *n* precum şi puterea *power*, repezintînd mărimile de ieşire din blocul M2:

$$n=\frac{\omega}{2*\pi} \quad \text{respectiv} \quad power=Tem*\omega_1 \qquad \text{(r6.12)}$$

Sub-blocul M2 din cadrul fişierului .csl generat, pe baza ecuațiilor (r6.6) – (r6.12) este următorul:

```
ACSL Block: M2
! Last Change: 15:26:10 05/10/97
! Description:
! Input Ports:
!    Port Name: voltage
!    Connected Variable: u
!    Port Name: freq
!    Connected Variable: f
!    Port Name: torque
!    Connected Variable: tpump
! Output Ports:
!    Port Name: speed
!    Connected Variable: n
!    Port Name: power
!    Connected Variable: power
!-------------------------------------------------------------------------
--
CONSTANT WeM2r1=0.048
CONSTANT WeM2b=0.1
CONSTANT WeM2r21=0.018
CONSTANT WeM2r22=0.010
CONSTANT WeM2jm=10.0
CONSTANT WeM2r23=0.053
CONSTANT WeM2pi=3.14
CONSTANT WeM2omega0=314.0
CONSTANT WeM2l1=0.00048
CONSTANT WeM2m1=3.0
CONSTANT WeM2l2=0.00048

WeM2omega1  = 2.0*WeM2pi *WeMCVSD2f
WeM2s =bound (0.0,1.0,(WeM2omega1  - WeM2omega )/(WeM2omega1
+0.00001))
WeM2R2s  = (WeM2R21 /(WeM2s +0.000001))+WeM2R22 +WeM2R23 *WeM2s
WeM2X1  = 2*WeM2pi *WeMCVSD2f *WeM2L1
WeM2X2  = 2*WeM2pi *WeMCVSD2f *WeM2L2
WeM2Tem  = ((WeMCVSD2v **2)/WeM2omega1 )*
       ((WeM2m1 *WeM2R2s )/(((WeM2R1+WeM2R2s )**2+(WeM2X1
+WeM2X2)**2)))
WeM2domega  = (WeM2Tem -WeP1torque -WeM2b *WeM2omega )/WeM2Jm
omega = integ(domega,omega0)

WeM2n  = WeM2omega /(2.0*WeM2pi )
WeM2power =WeM2Tem *WeM2omega1
```

Se observă în acest caz cît de complicate sunt uneori ecuațiile care descriu modelul matematic. Practic, modelul de mai sus a fost generat (adaptat) pe

baza unui model general preluat din cadrul unei biblioteci care conține diferite modele matematice pentru cele mai variate tipuri de componente.

Blocul care se referă la modelarea motorului cu turație variabilă M2, aşa cum apare în cadrul editorului grafic **ACSL/Graphic Modeller** este ilustrat în Figura 31.

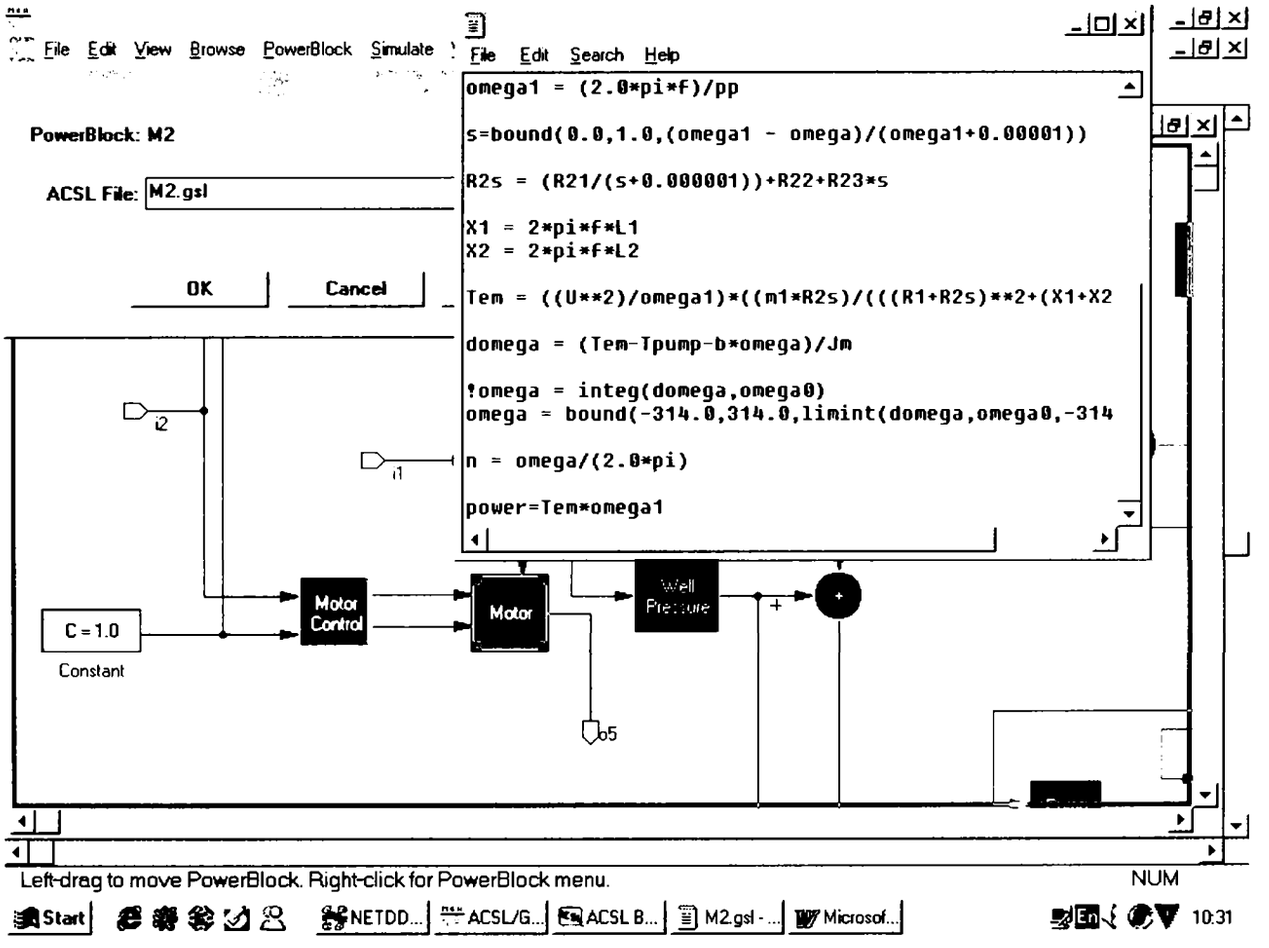


Figura 31: Blocul pentru controlul motorului M2
cu turație variabilă

c). Modelarea schimbului de căldură pe schimbătoarele din punctul termic

Blocul în care se realizează modelarea schimbului de căldură din punctul termic cuprinde un submodel pentru fiecare schimbător (s-a considerat cîte un schimbător generic pentru fiecare circuit, urmînd ca numărul real de schimbătoare să fie reflectat de suprafața de schimb de căldură). Fiecare schimbător are trei mărimi de intrare corespunzătoare celor trei porturi de intrare. Portul *i2* este însă un port prin care intrã o mărime vectorizată, formată din douã componente: cele douã debite a celor doi agenți termici respectiv

primar q_t şi secundar q_s. Acest vector este creat în blocuri de tip *Vectorize*, predefinit de ACSL.

S-a folosit acest artificiu pentru a reduce numărul de porturi şi cabluri care intră în blocul *het2*; sunt însă şi situaţii în care acest mod de lucru, prin vectorizarea unui set de variabile, este extrem de util, favorizînd un ulterior calcul algebric matricial.

Pe lîngă cele două debite mai avem ca mărimi de intrare entalpiile la intrare a celor două fluide: h_{prev} respectiv h_{sprev} . Entalpia de intrare a agentului este o funcţie de temperatura fluidului respectiv. Pentru circuitul de apă termală, entalpia masică a fost calculată succesiv în blocurile energetice de la rezervor şi de la conducta de alimentare. Astfel, în sursa pentru sub-blocul *het2* din cadrul fişierului .csl generat se regăsesc notaţiile prezentate mai sus cu două aceste excepţii: h_{prev} care se referă prin `PuPiEnen10h` şi respectiv h_{sprev} care se referă prin `HeHeCwcwt1h`. Acest lucru se datorează faptului că acestea sunt calculate în cadrul altor blocuri, după cum a fost precizat anterior, şi s-a dorit păstrarea convenţiei de notare.

Mărimile de ieşire ale blocurilor celor douã sunt temperaturile agenţilor primari t_t şi secundari t_s. Ele sunt obţinute din modelul de calcul prezentat în continuare. Astfel, pentru circuitul primar [Pop86]:

$$dh = (q_t\,(h_{prev} - h) - h_{tran})/(1000 * r_{oh} * v_h) \qquad \text{(r6.13)}$$

unde: r_{oh}, v_h constante care ţin cont de schimbul de căldură şi de cantitatea de lichid din schimbătoare
h_{tran} reprezintă cantitatea de căldură cedată de fluidul primar celui secundar
h entalpia fluidului primar

Prin integrare rezultă:

$$h = \int_{hin} \frac{dh}{dt} \qquad \text{(r6.14)}$$

In mod analog se pot scrie ecuaţii similare pentru circuitul secundar:

$$dh_s = (q_s\,(h_{sprev} - h) - h_{tran})/(1000 * r_{ohs} * v_{hs}) \qquad \text{(r6.15)}$$

unde: r_{ohs}, v_{hs} constante care ţin cont de schimbul de căldură şi de cantitatea de lichid din schimbătoare
h_{tran} reprezintă cantitatea de căldură primită de fluidul secundar de la cel primar
h_s entalpia fluidului secundar

rezultînd:

$$h_s = \int_{h\sin} \frac{dh}{dt} \qquad \text{(r6.16)}$$

Pe baza ecuațiilor (r6.14) şi (r6.16) se pot calcula temperaturile pe schimbătoare, care reprezintă mărimile de ieşire din acest bloc:

$$t_{prev} = \frac{h_{prev}}{4.18} \qquad t_{sprev} = \frac{h_{sprev}}{4.18} \qquad \text{(r6.17)}$$

$$tt = \frac{h}{4.18} \qquad ts = \frac{h_s}{4.18} \qquad \text{(r6.18)}$$

Aceste relații sunt cele uzuale pentru un schimbător de căldură cu plăci în contracurent. Sub-blocul *het2* din cadrul fişierului .csl generat, pe baza ecuațiilor descrise mai sus este următorul:

```
! ACSL Block: het2
! Last Change: 10:14:43 05/24/97
! Description:
! Input Ports:
!    Port Name: i1
!    Connected Variable: hprev
!    Port Name: i2
!    Connected Variable: qt
!    Connected Variable: qs
!    Port Name: i1a
!    Connected Variable: hsprev
! Output Ports:
!    Port Name: o1
!    Connected Variable: ts
!    Port Name: tt
!    Connected Variable: tt
!-------------------------------------------------------------------------
--
CONSTANT HeHeTrhet2rohs=1.0
CONSTANT HeHeTrhet2roh=1.0
CONSTANT HeHeTrhet2vhs=2.0
CONSTANT HeHeTrhet2vh=2.0
CONSTANT HeHeTrhet2hsin=50.0
CONSTANT HeHeTrhet2hin=300.0
HeHeTrhet2dh  = (HeHeTrVecqt *(PuPiEnen10h -HeHeTrhet2h )
              -HeHeTrhet2htran )/(1000.0*HeHeTrhet2roh *HeHeTrhet2vh )
HeHeTrhet2dhs  = (HeHeTrVecqs *(HeHeCwcwt1h -HeHeTrhet2hs )
              +HeHeTrhet2htran )/(1000.0*HeHeTrhet2rohs *HeHeTrhet2vhs
)
HeHeTrhet2h  = integ (HeHeTrhet2dh ,HeHeTrhet2hin )
HeHeTrhet2hs  = integ (HeHeTrhet2dhs ,HeHeTrhet2hsin )
```

```
HeHeTrhet2tprev  = PuPiEnen10h /4.18
HeHeTrhet2tsprev  = HeHeCwcwt1h /4.18
HeHeTrhet2tt  = HeHeTrhet2h /4.18
HeHeTrhet2ts  = HeHeTrhet2hs /4.18
```

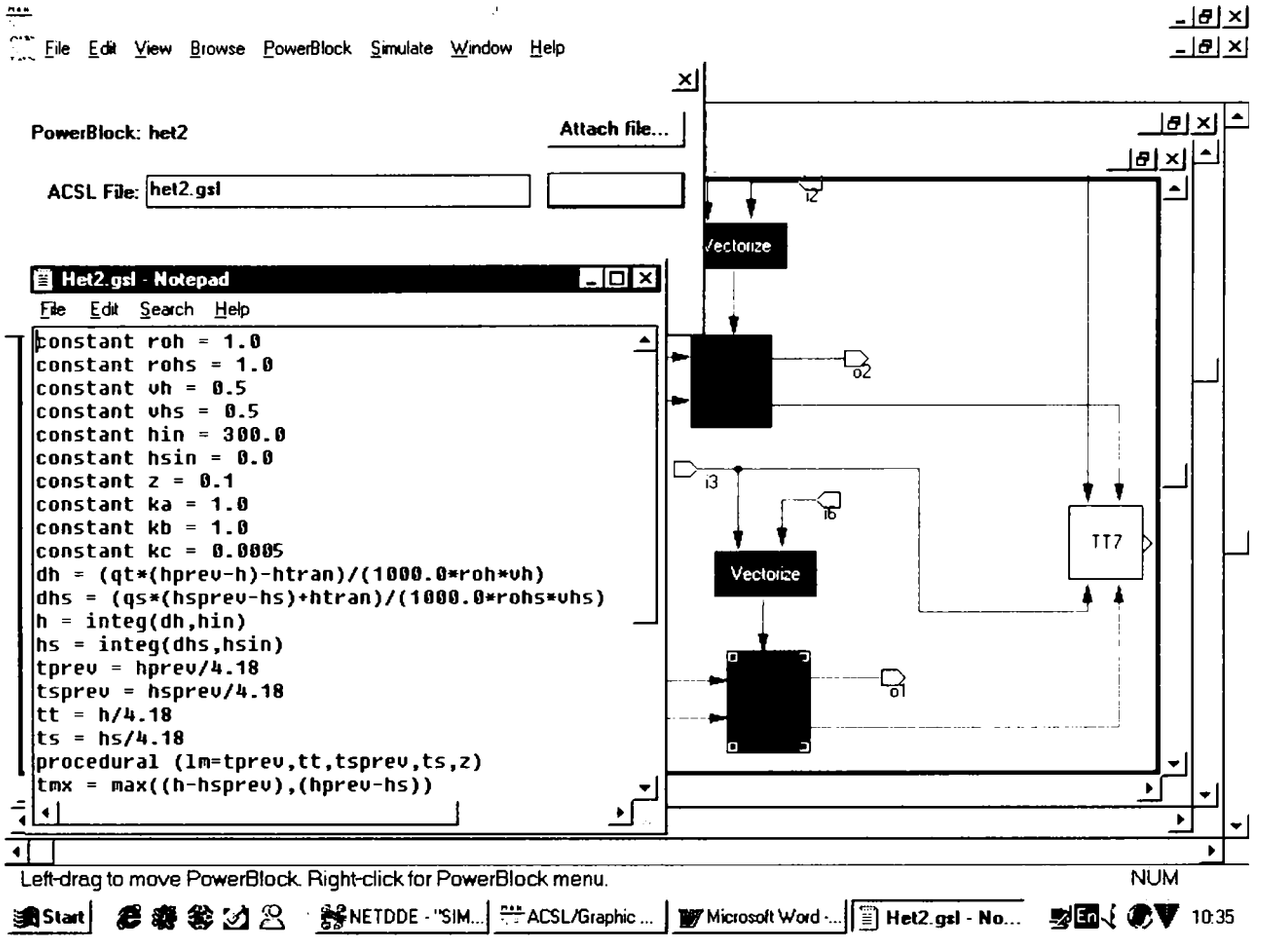


Figura 32: Blocul pentru modelarea transferului de căldură pe schimbătoarele din punctul termic

Blocul care se referă la modelarea transferului de căldură pe schimbătoarele din punctul termic, aşa cum apare în cadrul editorului grafic **ACSL/Graphic Modeller** este ilustrat în Figura 32.

In concluzie, se observă că în procesul de dezvoltare a modelului prescriptiv $\mathbf{M_p}$ fiecare element component din cadrul sistemului, indiferent că acesta reprezintă un element fizic sau de control, a fost modelat în mod analog exemplelor prezentate mai sus. Cu ajutorul facilităților puse la dispoziție de editorul grafic (paragraful 3.3.1.1.) s-a putut menține consistența structurilor modelului de-a lungul întregului proces de dezvoltare; de asemenea, în mod

automat a fost creată o bază de date cu explicații conținînd variabilele modelului, în scopul regăsirii acestora precum şi a unei mai bune documentări.

Un aspect foarte important l-a reprezentat posibilitatea testării separate a componentelor modelului; acest lucru a ajutat extrem de mult în partea de testare finală, de integrare, modelul de față fiind unul deosebit de complex.

Varianta grafică a modelului rezultat pentru întregul sistem se constituie în fişierul **oradea1.gm**. Varianta textuală al aceluiaşi model, este stocată în fişierul **oradea1.csl**. Acestea reprezintă practic modelul prescriptiv $\mathbf{M_p}$, formal, al sistemului în ansamblul său, şi care va fi utilizat în continuare în procesul de simulare.

Astfel, datorită utilizării software-ului **ACSL/Graphic Modeller**, simularea modelului $\mathbf{M_p}$ obținut este complet automatizată, fără a pune nici un fel de probleme deosebite. Acesta permite execuția modelelor astfel create, generînd, după cum s-a precizat şi anterior, automat cod în limbajul de simulare **ASCL** (fişierul **oradea1.csl**). Fişierul oradea1.csl care conține practic programul de simulare propriu-zis este în continuare este translatat în Fortran (**oradea1.for**), compilat şi linkeditat rezultînd în final fişierul executabil **oradea1.prx**, şi care conține practic modelul simulat propriu-zis.

6.2.2.2. Testarea modelului generat şi al simulării

Realizarea simulării a avut loc în paralel cu realizarea modelului $\mathbf{M_p}$, conform schemei prezentate în Figura 22. De asemenea, procesul de **verificare** a modelului a necesitat nu o dată revenirea (feedback) şi deci, menținerea consistenței între model şi simulare a reprezentat o problemă care a necesitat o atenție specială. In momentul în care a existat o simulare verificată, s-a trecut mai departe la procesul de **validare** al simulării. Aceasta s-a realizat în principal prin compararea rezultatelor obținute în procesul de simulare cu rezultate experimentale, culese din cadrul sistemului real.

Astfel, în procesul de validare al modelului, s-a urmărit în primul rînd investigarea proprietăților de bază ale modelului $\mathbf{M_p}$ generat precum şi măsura în care acesta respectă specificațiile date inițial. Au fost în acest sens depistate şi unele erori în procesul de proiectare, ceea ce a condus la revenirea în faza de dezvoltare a modelului (feedback), în conformitate cu schema de dezvoltare iterativă prezentată în Figura 6.

Pentru a putea executa modelul **oradea1.prx**, avînd în vedere complexitatea acestuia şi deci numărul mare de parametri care trebuiesc setați la valorile inițiale, au fost concepute diferite scenarii de execuție, sub forma unor fişiere scenariu. Un exemplu de astfel de scenariu este prezentat în Figura 33.

Se observă că, pe lîngă setarea la anumite valori inițiale a diferiților parametri utilizați în procesul de rulare, în cadrul acestui fişier se precizează şi alte constante cum ar fi: ***timpul total de simulare*** (Timelimit), ***intervalul de integrare*** (Cint), ***algoritmul de integrare*** utilizat (Ialg = Runge Kutta de ordinul 2 în acest caz), etc. Dintre aceştia, alegerea intervalul de integrare reprezintă cheia procesului de simulare: practic acesta dă cuanta de timp după care sunt reevaluate toate mărimile.

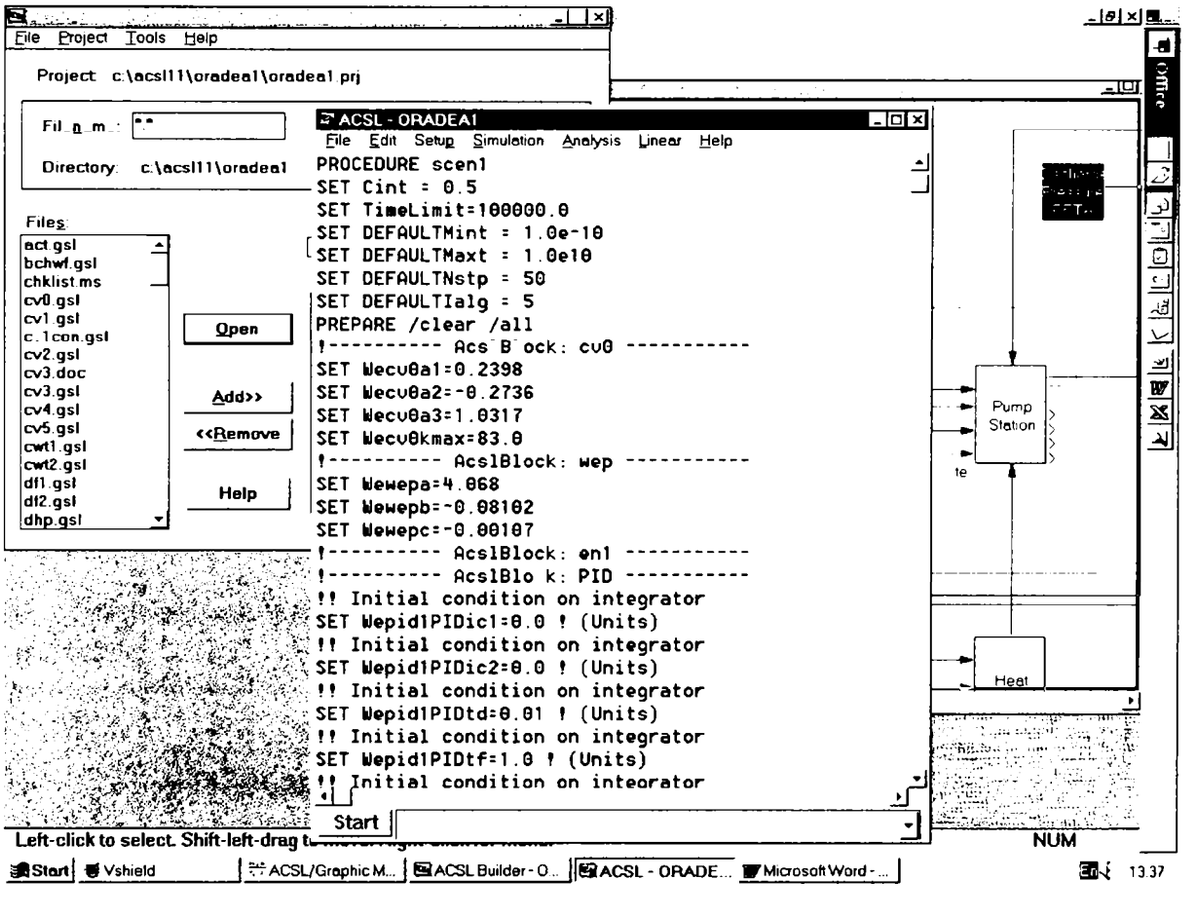


Figura 33: Scenariul de execuție scen1 (parțial)

Simularea performanțelor sistemului pe baza unor fişiere scenariu ca cel prezentat în Figura 33 este posibilă însă extrem de laborioasă, cu atît mai mult cu cît aceasta presupune rularea unui număr mare de simulări în condiții limită şi de încărcare maximă. Realizarea într-o manieră sistematică a cîte unui fişier-scenariu pentru fiecare situație care trebuie testată presupune existența a unui număr foarte mare de scenarii complexe, deoarece acestea trebuie să țină cont de faptul că o anumită cerință nu poate fi testată în izolare, ci într-o strînsă interdependență cu toate activitățile din cadrul sistemului. O alternativă originală care se propune în acest sens o reprezintă aşa numita ***testare interactivă***.

Descrierea de principiu (teoretică) a acestei metode de testare a fost prezentată în cadrul paragrafului 5.2.2.2.

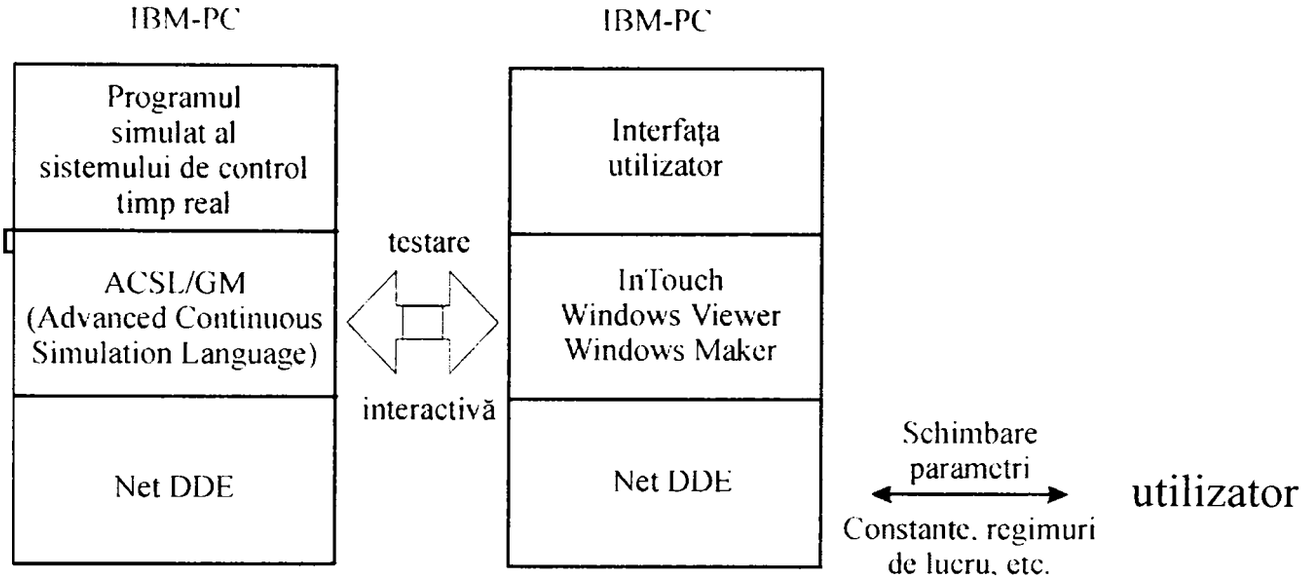


Figura 34: Schema de testare interactivă

Din punct de vedere practic, schema generală utilizată în procesul de testare interactivă este prezentată în Figura 34 şi ea a presupus realizarea următoarelor etape:

- modelul **Oradea1.prx** a fost lansat în execuție pe un calculator pe care rulează software-ul **ACSL**
- pe un al doilea calculator, a fost concepută o interfață utilizator a procesului (utilizînd software-ul **Wonderware InTouch**), prin intermediul căreia se va monitoriza/controla procesul simulat; această interfață a fost utilizată ulterior şi pentru monitorizarea şi controlul procesului real, prin schimbarea legăturilor, de la simulator către procesul controlat [Won94a]
- cele două calculatoare au fost legate în rețea (peer-to-peer), comunicarea între ele realizîndu-se utilizînd protocolului **DDE**, în cadrul căruia datele pot fi împărțite între **ASCL** şi **InTouch WindowViewer**, **ACSL** acționînd ca un server **DDE** care poate fi controlat din InTouch.

DDE este un protocol de comunicație **(Dynamic Data Exchange**) proiectat de Microsoft pentru a permite aplicațiilor Windows să transmită/primească date şi instrucțiuni unele de la altele. Acesta implementează o relație de tip client-server între două aplicații care rulează în mod concurent. Aplicația care

acționează ca server furnizează date şi acceptă cereri de la aplicațiile client. Unele aplicații, ca de exemplu InTouch sau Excel, pot fi atît server cît şi client.

Concret, comunicarea cu simulatorul a fost realizată în acest caz utilizînd aplicația **Allen Bradley Net DDE** [Won94b].

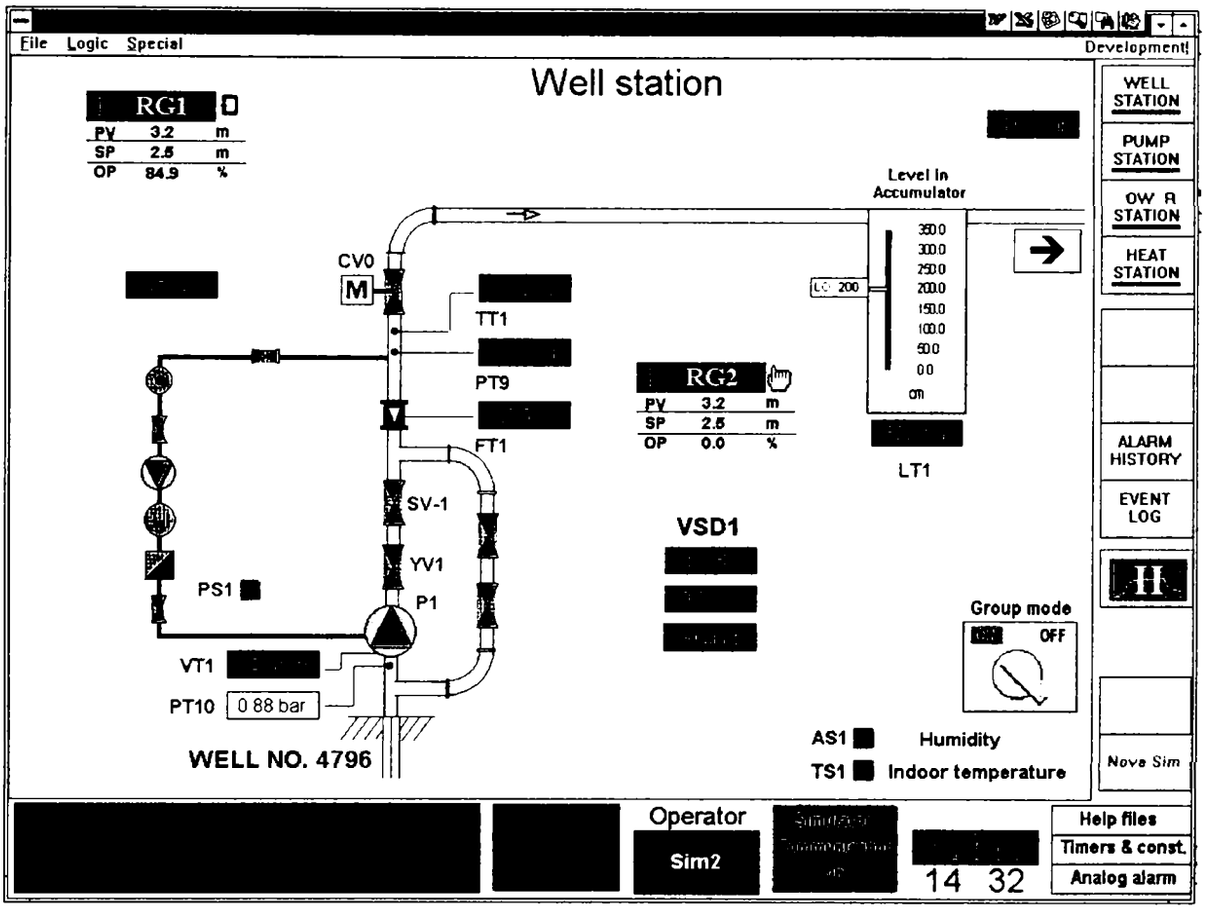


Figura 35: Interfața grafică pentru testarea interactivă a stației sondei (simulatorul)

Modalitatea de în care a fost realizată efectiv interfața utilizator utilizînd software-ul **Wonderware InTouch** este prezentată în paragrafului 6.4.2. In procesul de testare interactivă, utilizînd interfața InTouch accentul s-a pus pe simularea performanțelor sistemului, în scopul obținerii unei imagini cît mai exacte asupra dinamicii acestuia in general, avînd în vedere că modelul simulat $\mathbf{M_p}$ (verificat) se comportă în mod identic cu procesul controlat, comunicînd date către interfața utilizator şi totodată recepționînd date prin intermediul acesteia.

Astfel, pornind de la un scenariu unic, inițial, utilizatorul poate schimba în mod interactiv parametri de rulare (constante generale, set-point-uri, valori limită, constante PID [TL92], etc.) pentru a studia efectele acestora în sistem.

Un exemplu de fereastră din cadrul interfeței utilizator prin intermediul căreia s-a realizat testarea interactivă pentru stația sondei este prezentată în Figura 35. Se observă realizarea legăturii on-line cu simulatorul (**Simulator Communication On**). De asemenea. Regulatorul RG1 este pus pe auto, iar RG2 pe manual. Sunt de asemenea vizibile toate valorile mărimilor de interes: TT1, PT10, PT9, FT1, parametri regulatoarelor, modul de operare (grup on), etc.

Astfel, pentru ca testarea interactivă să decurgă într-o manieră sistematică, au fost urmate următoarele etape:

- ***testarea sistemului în condiții de funcționare normale***: adică au fost încercate toate operațiile posibile asupra sistemului; de exemplu: închidere/deschidere robinete, pornire/oprire motoare, comutarea în regim manual/automat, schimbarea diferiților parametri pentru regulatoare, schimbarea diferitelor constante, etc., urmărind de fiecare dată dacă reacția sistemului este cea dorită
- identificarea tuturor ***situațiilor limită*** precum şi a celor de ***încărcare maximă*** posibile; de exemplu: supraîncălzirea pompei P1 care necesită oprirea de urgență a acesteia sau existența unui nivel prea mare (alarma HH) a apei în rezervor
- identificarea tuturor ***situațiilor de defecțiune*** care pot apărea; de exemplu: imposibilitatea de a opera robinetul CV0, sau, mai grav, necesitatea opririi şi ulterior a repornirii sistemului în condiții de siguranță, datorită unor defecțiuni apărute şi care nu pot fi remediate cît timp sistemul funcționează
- ***stabilirea modalității efective de testare*** a fiecărei situații în parte, adică a valorilor care trebuiesc modificate interactiv astfel încît să se ajungă în situația respectivă şi ***realizarea testării*** in conformitate cu aceasta
- identificarea tuturor activităților cu deadline ferm şi determinarea **timpului aproximativ de execuție** pentru acestea, în toate situațiile posibile; în acest scop s-a realizat un număr mare de simulări, pentru fiecare activitate în parte, micşorînd din ce în ce mai mult intervalul de simulare (**Cint**), pînă în momentul în care timpul de execuție obținut s-a încadrat în limitele impuse de cerințe
- determinarea **intervalului de simulare Cint minim**, pentru care sunt satisfăcute toate cerințele temporale; valoarea acestui interval a constituit punctual de plecare în procesul ulterior de implementare. In cazul aplicației de față, **Cint minim** obținut a fost de **0.5 s**, suficient pentru a fi satisfăcute toate cerințele temporale ale aplicației, sistemul fiind din acest punct de vedere un sistem destul de lent.

Pe lîngă etapele de mai sus, în procesul de simulare a fost posibilă urmărirea grafică a evoluției în timp ale anumitor mărimi de interes, în scopul ajustării parametrilor regulatoarelor utilizate (constantele de proporționalitate, integrare şi derivare conform celor prezentate în paragrafului 6.2.2.1., pentru a se asigura o mai bună stabilitate în procesul de control [Dor92]. Un exemplu de astfel de grafic obținut în urma simulării modelului Oradea1 este ilustrat prin intermediul
Figurii 36. In cadrul acestuia, au fost repezentate grafic cîteva mărimi de interes, şi anume LT1, PT10 şi respectiv FT1 (pentru stația sondei), de-a lungul unei simulări care a durat 20.000 s.

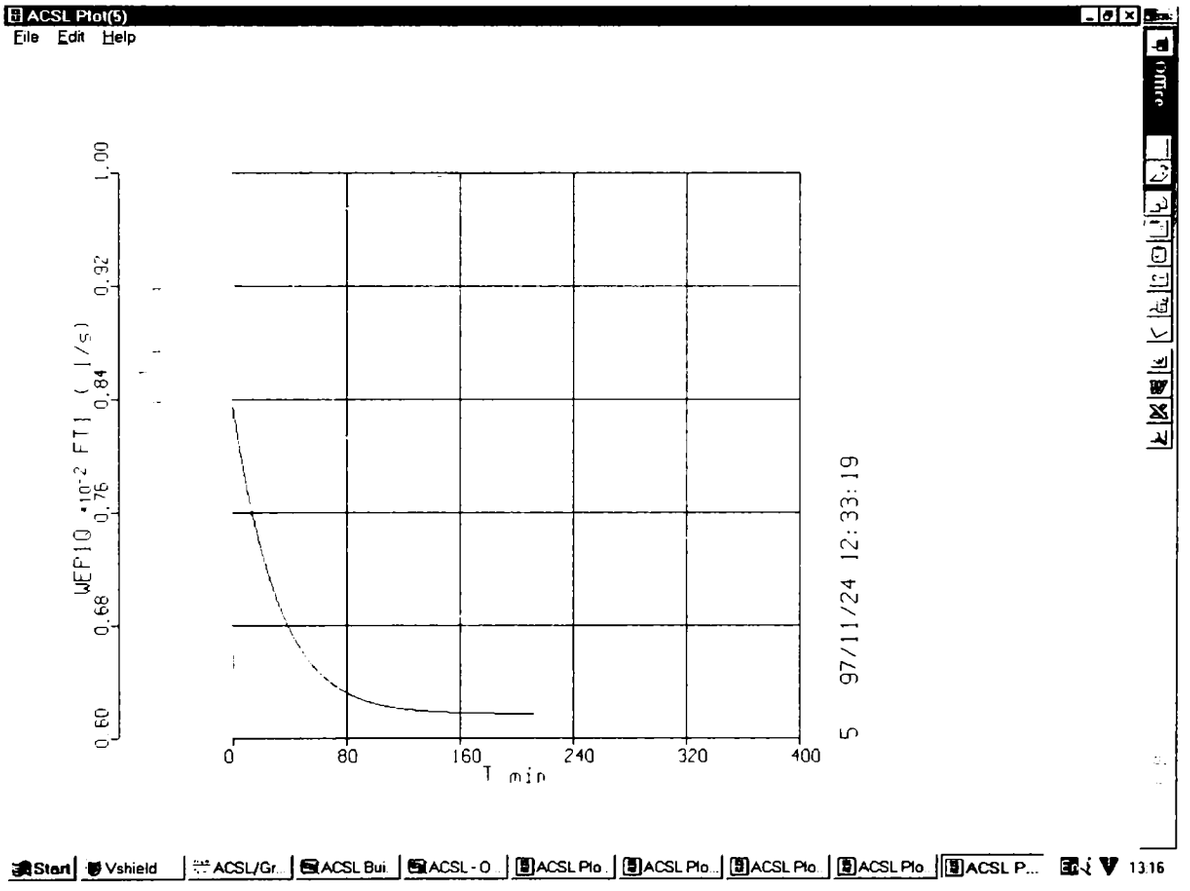


Figura 36: Evoluția mărimilor LT1, FT1 şi PT10 rezultată în urma simulării

6.3. Proiectarea software/hardware

În cadrul acestei etape a fost stabilită configurația generală hardware şi software a sistemului în cauză. Tot aici s-a stabilit care dintre funcțiile necesare trebuie realizate prin software şi care sunt cablate hard.

6.3.1. Proiectarea software

Din analiza structurii generale a software-ului de control timp real prezentat în Figura 25, precum şi a rezultatelor obținute în urma procesului de simulare a rezultat că software-ul de control din PLC trebuie să realizeze în principal următoarele funcții:

- achiziția, monitorizarea şi/sau controlul operațional al diferiților parametri: LT1, PT1, TT1, FT1, etc. în cazul controlului, acesta trebuie realizat utilizînd în acest scop diferite dispozitive: valve, comutatoare, motoare, variatoare de turație, etc.
- tratarea situațiilor de alarme inclusiv oprirea/pornirea sistemului
- trecerea sistemului din regim automat în regim manual şi invers
- gestionarea bazelor de date cu semnalele de I/O, alarme, etc.
- comunicarea cu interfața utilizator
- alte calcule intermediare (control statistic, evoluție în timp, etc).

Pentru dezvoltarea programului din PLC s-a ales mediul de programare **Advanced Programming Software (APS)** [Roc94b][Roc94c], care permite realizarea de programe pentru controller-ele din gama **Allen Bradley** [Roc94a] [Roc95]. Programele dezvoltate utilizînd APS sunt construite pe o structură modulară, fiind împărțite în mai multe blocuri funcționale care sunt văzute ca şi subrutine apelate dintr-un program principal, conform structurii generale prezentate în Figura 19.

Avantajul subrutinelor rezultă bineînțeles într-o minimizare a efortului de programare şi realizarea unor programe mai uşor de urmărit; din punct de vedere al proiectării, existența subrutinelor dă posibiliatea păstrării structurii ierarhice a strategiei de control, aşa cum a fost dezvoltată în cadrul modelului simulat, modulele din cadrul ierarhiei de control $\mathbf{C_p}$ putînd fi **mapate unu-la-unu** către module (subrutine, blocuri) din cadrul software-ului de control. De asemenea, au putut fi concentrate în subrutine separate prelucrări pentru subsisteme asemănătoare (sau chiar identice), în măsura posibilităților, sub o formă paramerizată.

Dacă ne referim la clasificarea făcută în paragraful 4.1.3, **APS** face parte din categoria de software-uri dedicate pentru controlul industrial, de tipul limbajelor specializate. Programarea propriu-zisă a software-ului de control a fost realizată direct pe PC-ul pe care rulează şi interfaţa utilizator; după ce programul a fost realizat şi testat, a fost ulterior descărcat (downloaded) către PLC.

Concret, pentru aplicaţia dată, din analiza specificaţiilor am alcătuit o structură de module pentru programul de control, $\mathbf{M_{impl}}$, utilizînd tehnica top-down, pe baza modelului $\mathbf{M_p}$, astfel:

- au fost extrase din modelul $\mathbf{M_p}$ numai porţiunile care se referă la subsistemele de control operaţional şi de tratare a alarmelor; modulele din cadrul modelului de implementare $\mathbf{M_{impl}}$ au fost alcătuite cu păstrarea structurii generale a codului, şi, după caz, completate cu noi module. Un exemplu în acest caz poate constitui modul în care s-a realizat semnalarea alarmei la PS2: presiune de absorbţie prea mică. Astfel, codul ACSL din cadrul modelului $\mathbf{M_p}$ care tratează secvenţa respectivă este următorul (blocul Pump Station `Pu`, sub-blocul `ps2`):

```
! ACSL Block: ps2
! Last Change: 12:14:15 05/17/97
! Description:
! Input Ports:
!    Port Name: i
!    Connected Variable: Tatl1p
! Output Ports:
!    Port Name: o
!    Connected Variable: o
!-------------------------------------------------------------
------------
CONSTANT Pups2imin=0.15
if(Tatl1p .lt.Pups2imin )then
Pups2o  = 1
else
Pups2o  = 0
end if
```

Practic, în cazul în care valoarea presiunii *p* (Tatl1p) este prea mică, ieşirea *o* (Pups2o) este setată la valoarea 1, altfel este setată la valoarea 0. Testarea valorii lui o (Pups2o) se face ulterior, într-un alt bloc. Secvenţe de genul celor de mai sus sunt practică transpuse identic în modelul de implementare $\mathbf{M_{impl}}$; singura diferenţă constă în schimbarea numelor variabilelor utilizate prin realizarea corespondenţei cu cele care vor fi utilizate în cadrul programului propriu-zis din cadrul PLC-ului

- după cum a fost precizat şi mai sus, a fost realizată corepondența tuturor variabilelor din baza de date a modelului $\mathbf{M_p}$ care se referă la I/O precum şi cele care sunt utilizate în cadrul strategiei de control cu variabilele utilizate în cadrul PLC-ului, şi respectîndu-se convențiile de notare pentru acesta
- pentru fiecare modul în parte au fost stabilite apoi locațiile de memorie corespunzătoare diferitelor variabile utilizate (utilizarea memoriei): I/O, timer-e, counter-e, variabile întregi şi binare, etc., fiecare locație fiind comentată pentru a fi mai uşor utilizată în program

Harta memoriei corespunzătoare I/O rezultată în urma acestei faze a fost utilizată pentru realizarea efectivă a structurii hardware: pe baza acesteia s-au realizat legăturile fizice cu senzorii şi dispozitivele de control. De exemplu, alocarea intrărilor, aşa cum este realizată pentru aplicația dată, este prezentată în cadrul Figurii 37 (o porțiune).

PLC-500 LADDER Oradea university PLC program

PLC-500 Data Base Form

RAFHNNUN HF

Data Base Form Sorted by:Address Page:00005 16:33 05/3

I:18.2	CV0_FB	CONTROL VALVE	POSITION			
I:18.3	PT9	WELL DISCHARGE	PRESSURE			
I:21/0	P3_OK	VSD2 READY				
I:21/1	P3_RUN	VSD2 RUNNING				
I:21/2	P3_FA	VDS2 COMMON	FAULT			
I:21/3	P3_ON	MOTOR RUNNING	POWER RELAY ON			
I:21/4	PS2_3	LOW SUNCTION	PRESS. F/BOOST			
I:21/5	P3_TD	THERMAL SWITCH	OFF PUMP OUT OF	OPERATION		
I:21/6	P3_OH	THERMISTOR IN	MOTOR			
I:21/7	P3_OL	OVERLOAD AND	SHORT CIRCUIT	TRIP		
I:21/8	P4_ON	MOTOR RUNNING	POWER RELAY ON			
I:21/9	P4_TD	THERMAL SWITCH	OFF PUMP OUT OF	OPERATION		
I:21/10	P4_OH	THERMISTOR	IN MOTOR			
I:21/11	P4_OL	OVERLOAD AND	SHORT CIRCUIT	TRIP		
I:21/12	PFS1	PHASE FAILURE	IN PANEL A3			
I:21/13	RL1	REMOTE LOCAL	OPERATION IN	PANEL A3		
I:21/14	ES1	EMERGENCY STOP	IN PALEL A3			
I:21/15	TS2	INDOOR TEMP.	TO HIGH IN	PUMP STATION		
I:21/16	AS2	AMBIENT HUMIDIT	TO HIGH IN	PUMP STATION		
I:21/17		SPARE				
I:21/18		SPARE				
I:21/19		SPARE				
I:21/20		SPARE				
I:21/21		SPARE				
I:21/22		SPARE				
I:21/23		SPARE				
I:21/24		SPARE				
I:21/25		SPARE				
I:21/26		SPARE				

Figura 37: Harta memoriei referitoare la alocarea intrărilor (o porțiune)

Posibilitatea de a vizualiza şi afişa modul de utilizare al memoriei cu ajutorul software-ului APS a permis alocarea într-o manieră ordonată a semnalelor în baza de date, urmărirea locațiilor neutilizate în vederea expansiunii ulterioare, evitarea suprascrierii. Avînd în vederea cantitatea mare de semnale utilizate, şi, în consecință, şi de variabile de memorie corespunzătoare, o astfel de facilitate este deosebit de utilă în procesul de programare.

Un calcul estimativ realizat în această faza preliminară de proiectare a software-ului a indicat faptul că este nevoie de aproximativ 40 intrări analogice, 160 intrări digitale, 4 ieşiri analogice şi respectiv 40 ieşiri digitale. Evident, aceste cifre au fost, în această etapă, orientative, fiindcă trebuie ținut cont şi de o eventuală expansiune (între anumite limite) a sistemului; se poate observa în acest sens că harta memoriei prezentată în Figura 37 conține şi poziții libere.

De asemenea, din analiza specificațiilor şi în urma datelor culese în procesul de simulare, a rezultat că un interval de scanare de aproximativ 0.5s este mai mult decît satisfăcător în cazul aplicației date. Aceste evaluări au stat la baza alegerii structurii hardware utilizate (procesoare, rețea de comunicație, etc.) precum şi a modalității de alocare a diferitelor module pe procesoare.

6.3.2. Alocarea modulelor pe procesoare (proiectarea hardware)

Avînd în vedere analiza realizată în cadrul procesului de simulare precum şi în cadrul celui de proiectare software, din punct de vedere al arhitecturii hardware s-a considerat satisfăcătoare alegerea unui sistem distribuit, format din trei noduri, după cum urmează:

- un nod constînd dintr-un **controller programabil PLC** pentru realizarea programului de control, echipat cu mai multe module de I/O prin intermediul cărora se realizează legăturile către procesul fizic, precum şi cu o sursă neîntreruptibiă de tensiune; subsistemele (modulele) de achiziție/monitorizare, control operațional şi de tratare a alarmelor sunt implementate în cadrul acestui nod (Figura 25)
- un nod constînd dintr-un **calculator PC** pentru realizarea interfeței grafice cu utilizatorul; subsistemele (modulele) de interfață cu utilizatorul, de control statistic, şi de istoric al evoluției sunt implementate în cadrul acestui nod (Figura 25)
- un nod suplimentar constînd dintr-un **modul DTAM Micro** [Roc95], care realizează o interfață cu utilizatorul în mod text, pentru cazul în care interfața grafică (PC-ul) se defectează; existența acestui nod este opțională, dar este benefică din considerente de siguranță. Practic acest modul dublează (într-o mai mică măsură) interfața utilizator de pe PC [Bar88]

Schema generală a sistemului rezultat este prezentată în Figura 38.

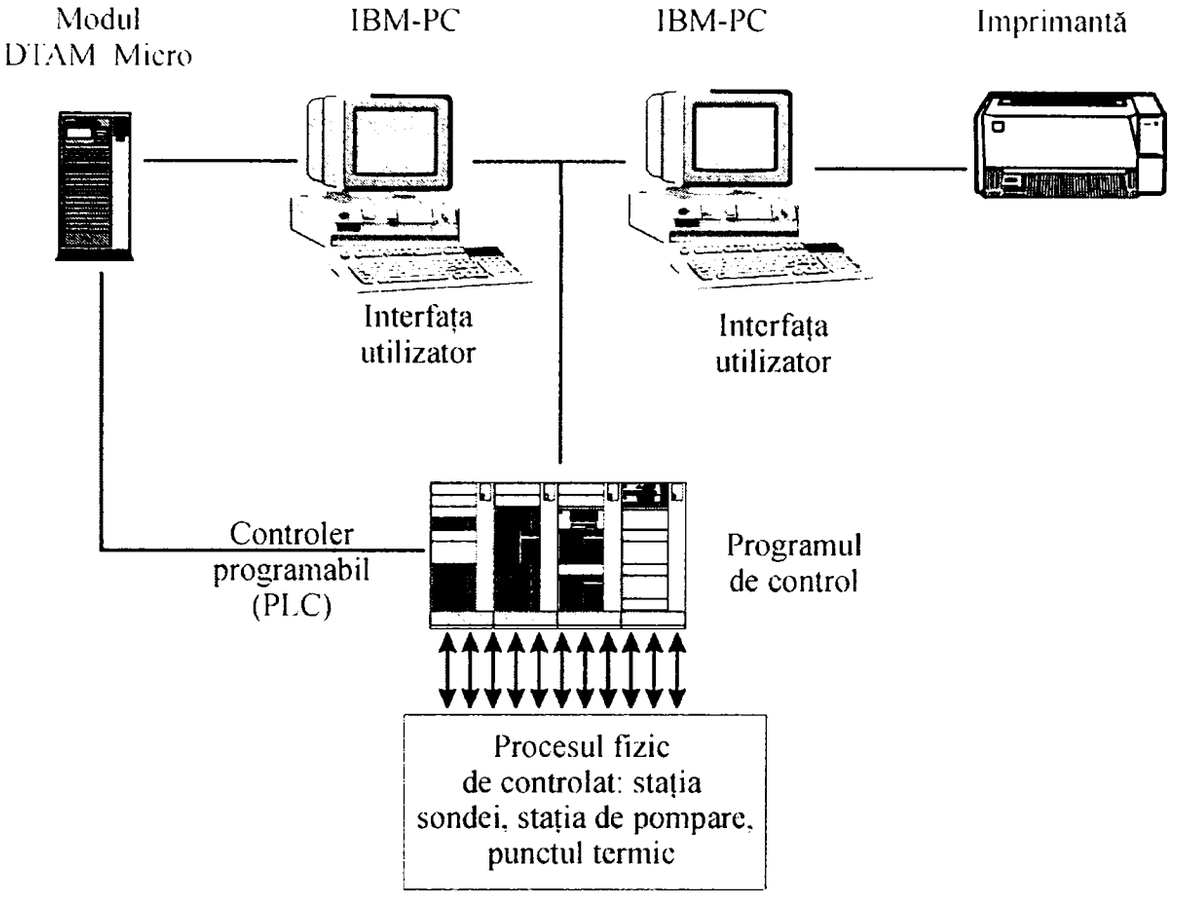


Figura 38: Shema sistemului de monitorizare şi control
a instalației geotermale de la Universitatea din Oradea

Astfel, pentru implementarea programului de control a fost ales procesorul **Allen Bradley SLC-5/03** din familia **SLC-500**, care nu este neapărat cel mai performant din gama firmei Allen Bradley, dar satisface pe deplin cerințele performanță/cost a aplicației date [Roc94a].

Familia **SLC 500** este o familie de controller-e mici, în general potrivite pentru aplicații dedicate, avînd însă o mare flexibilitate în realizarea de expansiuni ulterioare, incluzînd abilitatea de a comunica cu alte controller-e Allen Bradley de dimensiuni mai mari. Acestea oferă 20, 30 sau 40 de puncte de I/O discrete fixe, cu posibilități de de adăugare pînă la 64 puncte de I/O modulare pentru modulele de I/O discrete, analogice sau speciale.

Comunicarea între PLC şi PC a fost realizată pe linie serială; comunicarea între PLC şi modului DTAM Micro a fost realizată utilizînd o rețea de tipul DH-485, prin intermediul portului dedicat din cadrul PLC-ului.

Cîteva caracteristici ale procesorului SLC-5/03 care justifică alegerea făcută sunt următoarele [Roc94a]:

- structură modulară, formată din module funcționale care pot fi introduse într-un rack; numărul maxim de rack-uri care pot fi adresate este 3 (locale); modulele de I/O sunt astfel construite încît pot fi legate direct la traductoarele şi senzorii corespunzători, fără circuite intermediare de izolare şi condiționare a semnalelor
- oferă maxim 960 I/O dintre care 96 de semnale analogice
- memorie 12K pentru instrucțiuni, şi suplimentar încă 4K pentru date; după definitivarea programului acesta poate fi stocat într-o memorie single-chip EEPROM utilizînd facilitatea existentă în acest sens
- numărul de timer-e/counter-e este limitat numai de cantitatea de memorie disponibilă
- tabela de date este configurabilă de către utilizator
- timpul de scanare: 1ms/Kcuvînt; timpul de scanare pentru I/O : 0,225 ms
- furnizează un port RS-232 standard pentru comunicație serială (utilizat, în cazul de față, pentru comnicația cu PC-ul)
- furnizează un port DH-485 (utilizat în cazul de față pentru realizarea legăturii cu modulul DTAM).

Pentru a satisface necesarul de I/O pentru aplicația dată, precum şi pentru a prevedea posibilitatea unei ulterioare extensii, s-au considerat necesare următoarele module de I/O:

- 12 module analogice de intrare: furnizează 4 canale de intrare analogice ($\pm$10 V, $\pm$20 mA) pe 16 biți; nu necesită sursă de alimentare externă
- 1 modul analogic de ieşire: furnizează 4 canale de ieşire analogice (0 ... 20 mA) pe 16 biți; nu necesită sursă de alimentare externă
- 5 module de intrare digitale: furnizează 32 de canale de intrare digitale (24 V CC)
- 2 module de ieşire digitale: furnizează 32 de canale de ieşire digitale (10/50 V CC), dintre care unul este momentan neutilizat

Astfel, structura generală hardware a modulelor din PLC pentru aplicația dată este sintetizată prin schema din Figura 39:

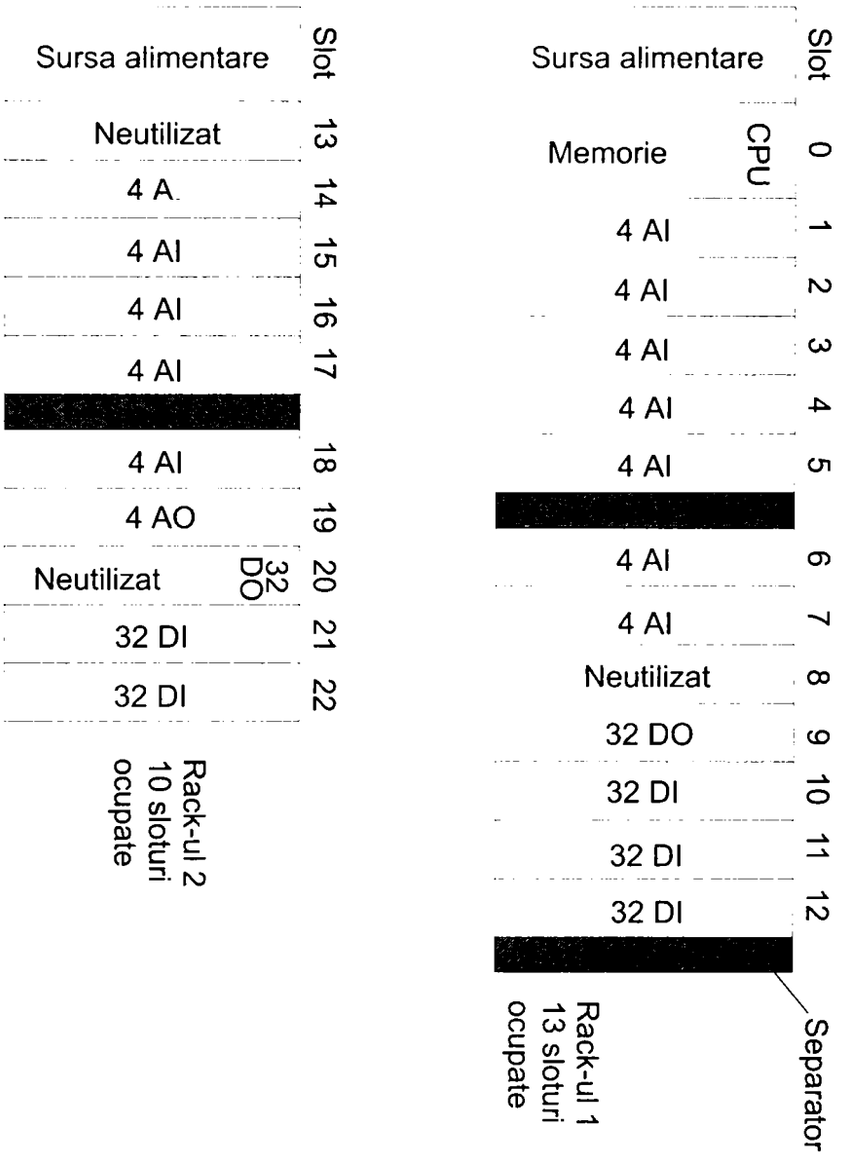


Figura 39: Structura generală hardware a modulelor de I/O din PLC

Calculatorul PC utilizat pentru realizarea interfeței cu utilizatorul a fost ales un Pentium la 100 MHz, care, în 1996, cînd am început proiectarea sistemului, reprezenta unul dintre cele mai puternice PC-uri de pe piață. Unul dintre avantajele separării interfeței cu utilizatorul de programul de control este că, pe de-o parte acesta este degrevat de realizarea unor prelucrări necesare procesului de afişare (ex: scalări), şi pe de altă parte, se pot astfel stoca mari cantități de date utilizînd hard-disk-ul PC-ului, date care pot fi utilizate ulterior pentru a urmări evoluția în timp a anumitor parametri (grafice, calcule statistice) [SMY00].

Acest lucru a fost realizat de asemenea prin intermediul interfeței utilizator, cantitatea de date disponibilă la un anumit dat nefiind limitată decît de spațiul disponibil pe disc. Tot în cadrul PC-ului au fost realizate şi anumite calcule (totalizări, scalări) necesare pentru a obține anumite date de ansamblu asupra procesului, programul de control din cadrul PLC-ului fiind astfel redus la minimum necesar.

Pentru cazul în care PC-ul care conține interfața grafică se defectează, am prevăzut modul DTAM Micro Operator Interface, cu ajutorul căruia se poate realiza monitorizarea şi controlul unui număr redus de parametri (cei care sunt absoult esențiali), spre deosebire însă de PC, în mod text (astfel, maximum 50 de ferestre sunt permise de memoria existentă: afişare date, alarme, configurare). Utilizarea acestui modul este opțională, dublînd (într-o mai mică măsură, evident), interfața grafică de pe PC [VMP92].

6.4. Implementarea şi testarea software-ului aplicației

In cadrul acestei etape a fost implementat efectiv programul de control cît şi interfața utilizator, pe baza modelelor dezvoltate mai sus.

6.4.1. Implementarea software-ului din PLC

Practic, pentru aplicația în cauză, programul din PLC a fost împărțit în 29 de module, conform listingului următor (preluat din APS):

```
      #0  System datastorage header]
      #1  Reserved area]
File #2  Main ladder program
File #3  Controlers in Pump station
File #4  Controlers in Well station
File #5  Motor start/stop in Heat station
File #6  Analog limits and transmitter faults
File #7  Mode selections and alarms for Heat station
File #8  Controlers in Heat station
File #9  Initialisation of all start conditions
File #10 Motor start/stop in Well station
File #11 General alarms for all stations
File #12 Setings in scada when group mode=off
         in Well station
File #13 Motor start/stop for pump station
File #14 Setings in scada when group mode=off
         in Heat station
File #15 Setings in scada when group mode=off
         in Pump station
File #16 Calculations of flow meter pulses
File #17 User settings in heat station for power up start
File #18 User settings in well station for power up start
File #19 User settings in pump station for power up start
File #20 Settins for stopping all motors in heat station
File #21 Settins for stopping all motors in well station
File #22 Settins for stopping all motors in pump station
File #23 Settings for starting heat station
         after power fault
File #24 Calculations of TT18 from TT17 for 24h average
File #25 File for calculating of TT17 average per hour
File #26 Maping of inputs and outputs to an address
File #27 Wind speed sensor calculations
File #28 Calculations of TT14 average per hour
File #29 Calculations of TT16 from TT14 for 24 hours
File #30 File for emulating outputs and inputs
```

Se observă existența unui număr destul de important de module: inevitabil, realizarea unui software robust conduce la programe complicate: statisticile

arată că doar 25% din software-ul propriu-zis tratează funcționarea normală, restul, de aproximativ 75% se ocupă cu tratarea situațiilor de alarmă (extreme), care, în general, apar destul de rar. Această protecție, cu inevitabila creştere a dimensiunilor programului, este însă necesară pentru toate sistemele care operează în timp real, şi în consecință, a fost utilizată şi în cazul de față.

Astfel, o atenție deosebită a fost acordată, în procesul de implementare, porțiunilor de cod care tratează anumite alarme, în special cele legate de factorul timp. De exemplu, după cum a fost prezentat în cadrul specificațiilor, în etapa de control algoritmic, pentru operarea valvei CV0 cerințele sună în felul următor: ***dacă CV0 dă eroare de limită de închidere pentru mai mult de t_{L1} secunde, se va opri operarea acesteia, şi se va trece regulatorul RG1 în regim manual, cu ieşire fixată***.

Pentru tratarea unei astfel de situații s-a utilizat în implementarea programului un timer, cu valoarea presetată egală cu limita de timp t_{L1}. În momentul în care timpul limită este atins, programul execută operațiile necesare (trece regulatorul RG1 în regim manual, cu ieşire fixată), şi, de asemenea, poziționează o variabilă de memorie prin intermediul căreia se va semnala alarmă către operator prin intermediul interfeței utilizator, astfel încît acesta să poată iniția anumite acțiuni în consecință, dacă este cazul. Întîrzierea maximă care poate apărea în tratarea alarmelor datorită utilizării abordării ciclice în procesul de implementare este de maxim două perioade de scanare (Figura 20), ceea ce, avînd în vedere că toate constantele de timp existente (t_{w*}, t_{L*}, etc.) sunt mult mai mari, poate fi neglijată.

Practic, în implementarea programului s-a încercat tratarea, într-o manieră asemănătoare, a tuturor situațiilor de acest gen; unele dintre acestea necesitînd intervenția operatorului, altele nu. O atenție deosebită a fost acordată de asemenea porțiunii de cod care tratează oprirea şi respectiv repornirea sistemului în condiții de siguranță, în urma unei căderi de tensiune. De asemenea, pentru o mai mare flexibilitate, toate constantele de timp precum şi alte constante utilizate în program sunt transmise prin intermediul variabilelor de memorie către interfața utilizator astfel încît operatorul să aibă posibilitatea de a le modifica, în anumite cazuri (dacă are un nivel de acces corespunzător).

Pentru a contracara, într-o oarecare măsură creşterea programului rezultată din aceste considerente, în procesul de implementare a modulelor am utilizat, pe cît posibil, un număr minim de instrucțiuni. Această tendință de a minimiza numărul de instrucțiuni, rezultînd un timp de scanare mai scurt, a fost însă utilizată doar în măsura în care ea nu afectatează claritatea programului.

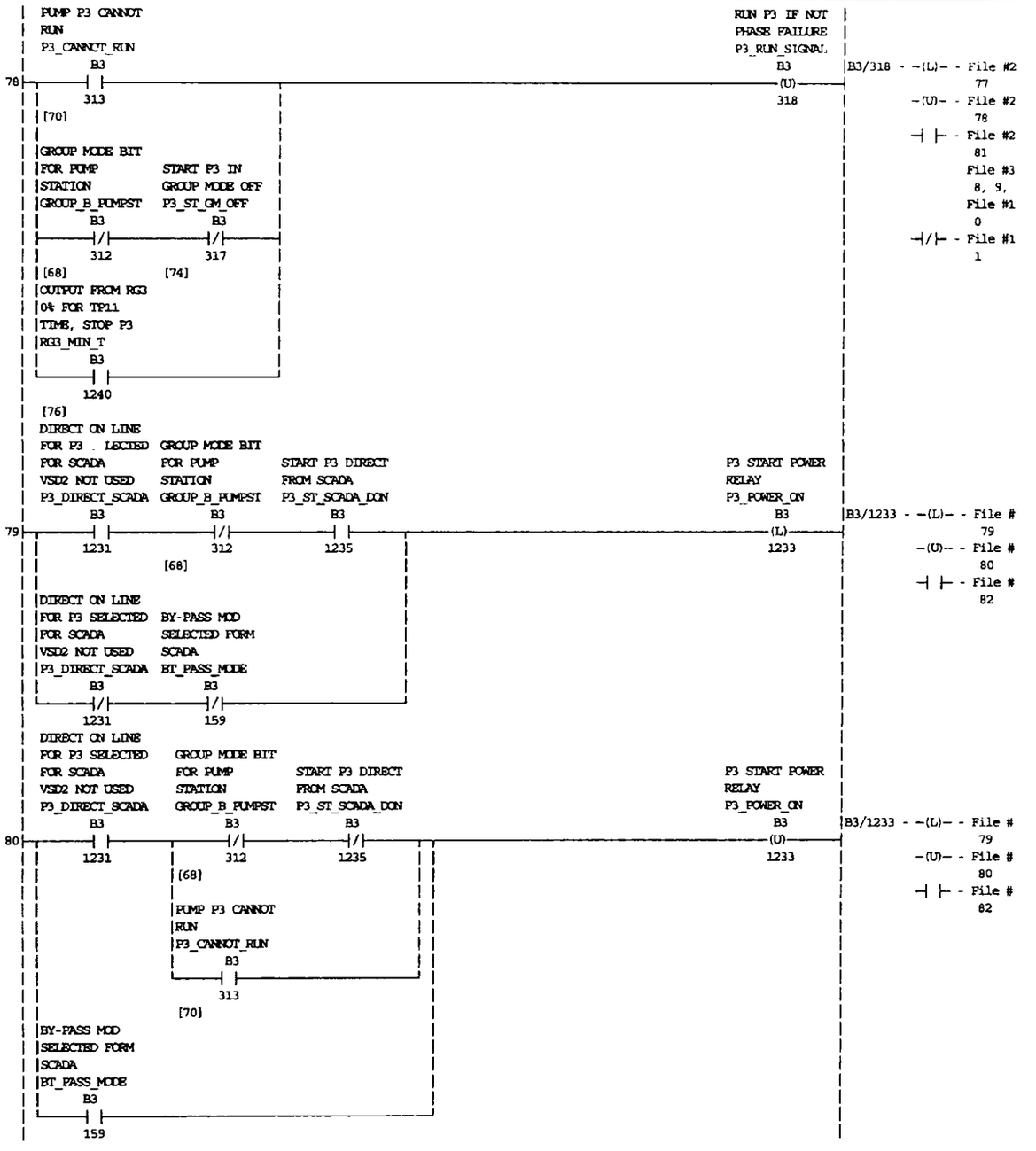


Figura 40: O porțiune din programul PLC (ladder logic)

Pentru a ilustra modul în care s-a realizat efectiv programul, în Figura 40 este prezentată o secvență din cadrul acestuia. De asemenea, pentru a avea o idee referitoare la dimensiunea programului rezultat în ansamblu, trebuie spus că acesta are, în varianta actuală, un număr de 29 de fişiere de program,

91 de fişiere de date, constă dintr-un număr de aproximativ 7000 de instrucţiuni (6828), şi prelucrează un număr de aproximativ 200 semnale de I/O, putînd fi considerat ca făcînd parte din categoria aplicaţiilor de dimensiuni medii. Perioada de scanare obţinută este de aproximativ 160ms, mai mult decît satisfăcătoare nevoilor aplicaţiei date.

6.4.2. Implementarea interfeţei utilizator

În ceea ce priveşte interfaţa utilizator, realizarea unei interfeţe conform standardelor actuale n-ar putea fi făcută fără unelte corespunzătoare. Tipice în acest caz sunt aşa numitele ***sisteme de management a interfeţelor utilizator***. Astfel de sisteme furnizează în general un set de obiecte de bază (icoane, meniuri, bare de defilare, etc.), care pot fi utilizate pentru particularizarea anumitelor interfeţe precum şi biblioteci de rutine şi obiecte predefinite. De asemenea, acestea mai furnizează şi o modalitate de a se lega la programul de aplicaţie (sau la simulator, după caz) corespunzător pentru a putea suporta intrări/ieşiri.

Astfel de unelte sunt în general alcătuite din două părţi: o ***componentă de dezvoltare*** şi o componentă ***run-time***. Componenta run-time facilitează realizarea legăturilor între structurile interne de date şi prezentarea externă a acestora. Componenta de dezvoltare este utilizată în faza de realizare a interfeţei, de obicei utilizînd un editor grafic în combinaţie (eventual) cu un limbaj de programare.

Pachetul **SCADA InTouch** [Boy93] **(Supervisory Control And Data Acquisition)** al firmei **Wonderware InTouch**, utilizat în cazul aplicaţiei date, face parte din categoria sistemelor de management a interfeţelor utilizator prezentate mai sus, cele două componente ale acestuia se numesc în acest caz **InTouch WindowMaker** (pachetul de dezvoltare) şi respectiv aplicaţia construită astfel va rula în **InTouch WindowViewer** (componenta run-time).

InTouch WindowMaker şi **InTouch WindowViewer** furnizează următoarele facilităţi [Won94a]:

- un editor grafic
- o serie de biblioteci grafice de obiecte
- posibilitatea de prezentare alarme
- posibilitatea de trasare de grafice pe baza datelor culese
- posibilitatea de a seta diferite nivele de securitate
- lucrul cu baze de date de pînă la 32.000 de elemente (tag-uri).

Utilizînd această interfață grafică, este permis operatorului, pe lîngă activitatea de monitorizare, să realizeze un control deosebit de complex al procesului, prin modificări de set-point-uri, închidere/deschidere valve, pornire/oprire motoare, etc., realizat de la distanță prin intermediul sistemului SCADA implementat. Conectarea cu programul de control din cadrul PLC-ului s-a realizat utilizînd **Allen Bradley Serial DDE Server** [Won90].

Structura interfeței dezvoltate se bazează pe noțiunea de fereastră care conține desenul schematic ale diverselor porțiuni ale sistemului. Astfel, pentru sistemul concret dat, în urma specificațiilor a rezultat clar (din partea referitoare la interfața cu utilizatorul, paragraful 6.2.1.2.) care sunt informațiile care trebuiesc afişate pe ecran; de asemenea, care sunt informațiile asupra cărora trebuie să se poată exercita controlul, etc. Fiecare din aceste informații au fost asociate cu fereastra corespunzătoare; de asemenea, legăturile între ferestre (dacă este cazul) au fost stabilite în această fază.

Principalele tipuri de ferestre concepute în acest caz sunt următoarele:

- ferestre pentru realizarea imaginii de ansamblu pentru fiecare subsistem în parte: stația sondei, stația de pompare, punctul termic
- ferestre pentru afişarea alarmelor şi atenționărilor
- ferestre pentru urmărirea variațiilor în timp a diferiților parametri (real-time trends, historical trends)
- ferestre pentru vizualizarea/modificarea stării regulatoarelor
- ferestre pentru vizualizarea/modificarea diferiților parametri de operare de către operator, etc.

Un exemplu al imaginii procesului în InTouch, şi care ilustrează subsistemul stația sondei este prezentat în Figura 41. Se observă, din această figură existența regulatoarelor RG1 şi RG2, sub forma unor dreptunghiuri în cadrul cărora apar trei valori: variabila de proces PV (controlată), set-point-ul SP (pre-setat), semnalul de ieşire dat de regulator OP, precum şi modul de operare al acestuia (MAN=manual, AUTO=automat). Acestea au fost create de o asemenea manieră încît utilizatorul poate schimba constantele regulatorului (constante PID), set-point-ul precum şi modul de operare al acestuia selectînd căsuțele corespunzătoare de pe ecran.

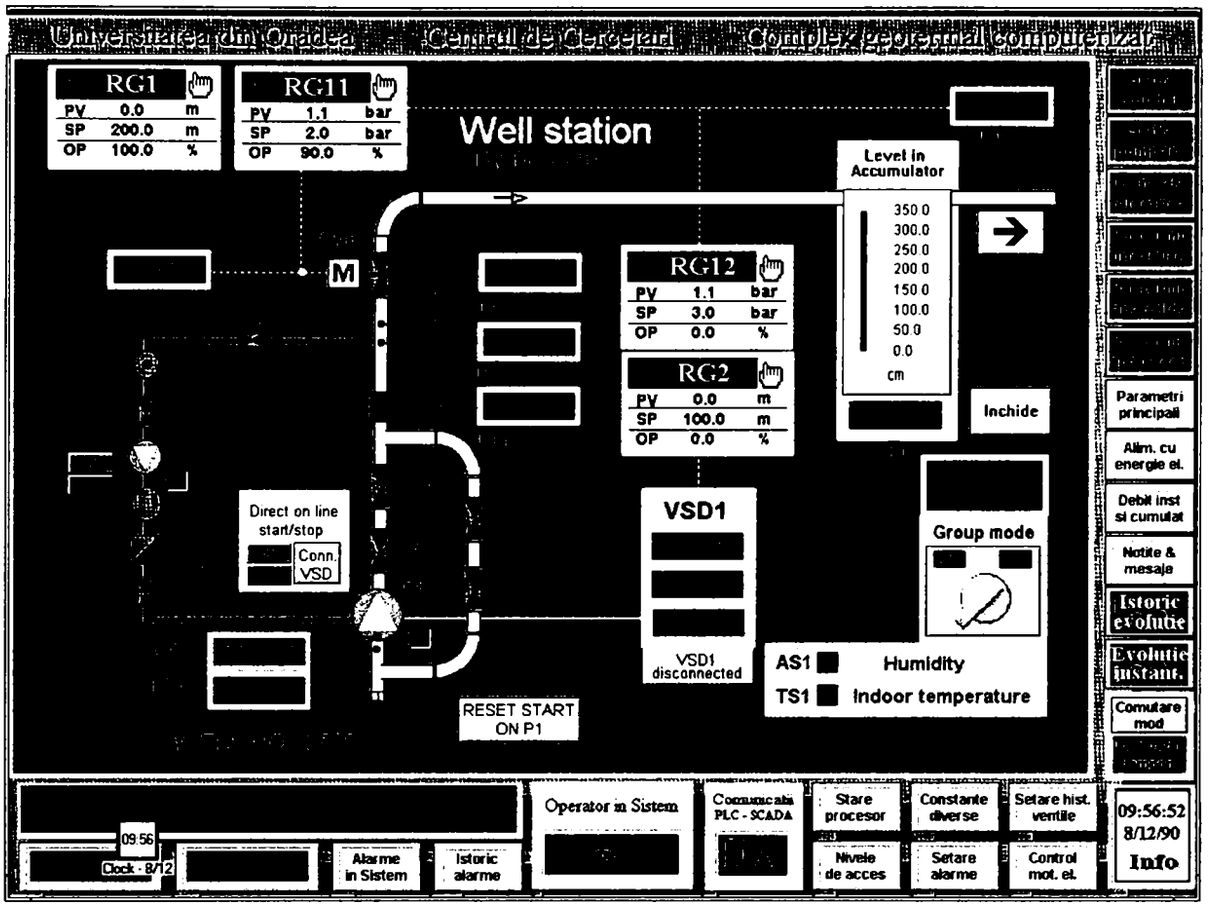


Figura 41: Fereastra corespunzătoare staţiei sondei

De asemenea, la realizarea acestor ferestre au fost stabilite anumite convenţii de reprezentare: de exemplu, pompele sunt reprezentate cu culoarea verde cînd sunt în funcţiune, gri deschis cînd sunt oprite dar sunt în stare de funcţionare, şi respectiv roşie cînd prezintă o defecţiune. Acelaşi lucru este valabil şi pentru variatoarele de turaţie VSD1 şi VSD2. Robinetele sunt reprezentate utilizînd simboluri speciale pentru marcarea poziţiilor "complet închis", "complet deschis", operare în direcţia de închidere, operare în direcţia de deschidere, operare cu deschidere fixă sau defecţiune. Staţia sondei poate fi trecută în modul grup off/on prin selectarea poziţiei corespunzătoare a comutatorului din cadrul ferestrei.

Fereastrele corespunzătoare staţiei de pompare şi respectiv punctului termic sunt prezentate în Figura 42 şi respectiv Figura 43. Se observă păstrarea aceloraşi principii de creare a intefeţei utilizator în cazul acestor ferestre, ca şi la staţia sondei.

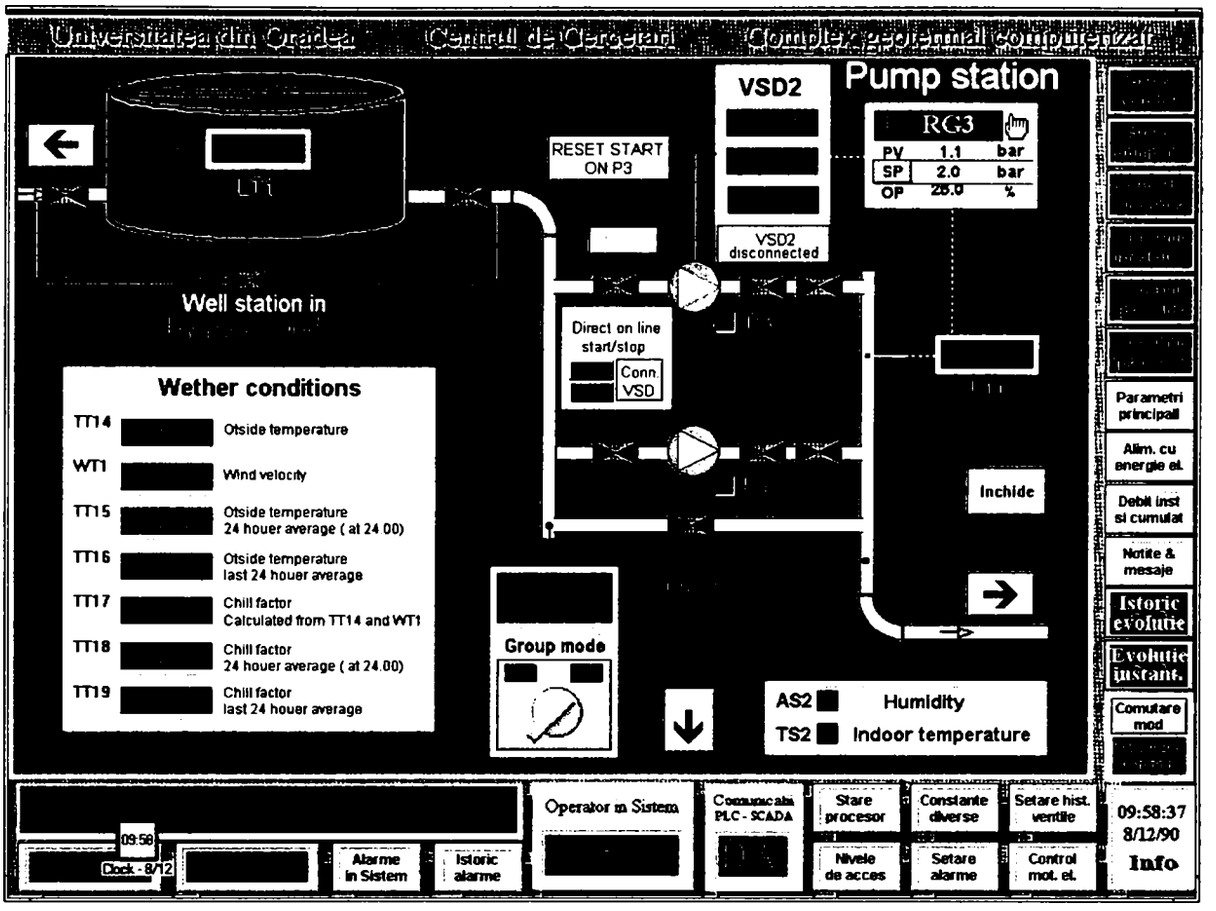


Figura 42: Fereastra corespunzătoare stației de pompare

Astfel, s-a urmărit la crearea interfeței utilizator, realizarea unor ferestre care să conțină cît mai multă informație, repartizată de o asemenea manieră încît să fie cît mai uşor de urmărit, dar şi ofere o cît mai are flexibilitate aplicației. Scalarea tuturor mărimilor afişate a fost realizată prin tot intermediul interfeței, şi de asemenea unele totalizări necesare pentru managementul procesului au fost realizate utilizînd macro-uri în Visual Basic.

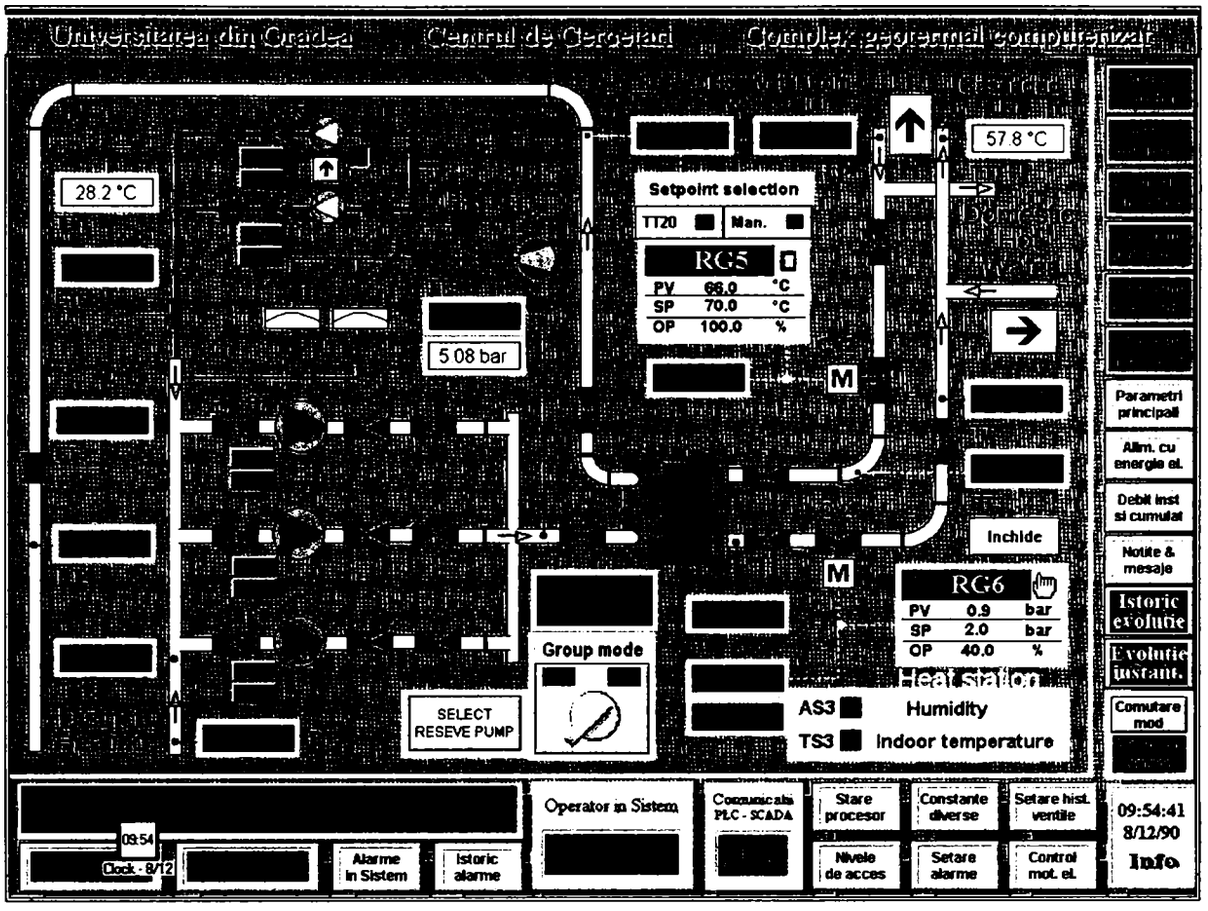


Figura 43: Fereastra corespunzătoare punctului termic

6.4.3. Testarea software-ului implementat

Procesul de testare a software-ului implementat a fost realizat în principal în două etape:

- ***testarea separată a modulelor*** din cadrul programului de control; în această fază, programul de control nu a fost conectat încă la procesul fizic. Practic, în cadrul acestei etape preliminare am folosit facilitatea PLC-ului Allen Bradley de a putea forța intrările binare la anumite valori, pentru a simula diferite situații. Pentru intrările analogice, am utilizat locații de memorie în locul intrării propriu-zise. Aceste setări au fost realizate prin accesul direct la tabela de date din PLC.
- ***testarea programului de control*** în ansamblul său, cu intrările şi ieşirile conectate direct la procesul fizic de controlat. S-a utilizat **testarea interactivă** în acestă etapă, prin intermediul aceleiaşi interfețe utilizator ca şi în cazul testării simulatorului. Practic, testarea de ansamblu s-a realizat în timp real, programul fiind conectat la procesul fizic controlat.

Trebuie menționat aici faptul că scenariile de test alese precum şi etapele corespunzătoare ***au fost aceleaşi*** cu cele utilizate în cazul modelului simulat, (prezentate în paragrafului 6.2.2.2.) datele rezultate fiind în permanență comparate cu cele obținute în urma simulării.

Spre deosebire însă de simulare, realizarea testării în timp real a sistemului a ridicat probleme deosebite legate de protecție (în general pentru testarea condițiilor limită). De asemenea, repetabilitatea unei anumite testări a fost greu de obținut, pe de-o parte datorită faptului că, sistemul fiind destul de complex, fiecare stare a acestuia depinde de o mare varietate de factori (nefiind un sistem concurent nu se pune problema nedeterminismului) şi pe de altă parte, datorită faptului că procesul fizic controlat este lent.

Se observă că, pe ansamblu, procesul de testare respectă maniera de testare dinamică pentru sisteme de control timp real prezentată în paragrafului 1.3.4.5., trecînd prin toate etapele acesteia. Astfel, toate testările pînă în acest moment au fost realizate utilizînd un model simulat; testarea finală a fost însă realizată direct pe procesul real.

6.5. Concluzii

In cadrul acestui capitol a fost prezentat un exemplu de aplicație timp real care a fost dezvoltată utilizînd metodologia propusă în cadrul capitolului 5. Se observă că etapele care au fost urmate sunt cele caracteristice procesului de dezvoltare a unei aplicații generale de timp real, conform schemei din Figura 7, şi anume:

- ***analiza şi specificarea cerințelor*** – în cadrul căreia a fost elaborat un document, informal, dar care sistematizează cerințele sistemului în ansamblul său (paragraful 6.1.)
- ***controlul algoritmic*** – în cadrul acestei etape s-au utilizat cu precădere tehnicile **ICA**, în special la dezvoltarea strategiei şi apoi a algoritmilor de control. Practic, această etapă s-a concretizat în dezvoltarea modelelor $\mathbf{M_d}$ şi respectiv $\mathbf{M_p}$ pentru sistemul în cauză, şi anume (paragraful 6.2.):
  - modelul $\mathbf{M_d}$, **informal**, care descrie sistemul într-o manieră ierarhizată, sub formă de schemă bloc; **ACSL/Graphic Modeller** reprezintă mediul integrat cu ajutorul căruia s-a realizat acest model
  - modelul $\mathbf{M_p}$, **formal**, care descrire sistemul sub forma unor ecuații matematice diferențiale neliniare; **limbajul de simulare ACSL** a fost utilizat pentru descrierea modelului $\mathbf{M_p}$ şi, în consecință, a fost posibilă validarea ulterioară a acestuia prin simulare
- ***proiectarea software/hardware*** – în cadrul acestei etape s-a stabilit, pe baza rezultatelor obținute în procesul de simulare, care sunt modulele software care trebuie implementate precum şi modul de alocare a acestora pe procesoare; structura generală hardware a sistemului a fost definitivată, principalul criteriu care a fost luat în considerare fiind cel al ***asigurării tuturor resurselor necesare*** (**resource adequacy**). Concret, structura sistemului s-a constituit dintr-un **controller programabil Allen Bradley** pentru realizarea programului de control propriu zis, un **calculator PC** pentru realizarea interfeței cu utilizatorul precum şi un **modul DTAM Micro** auxiliary (paragraful 6.3.)
- ***implementarea software-ului şi testarea*** – implementarea software-ului a fost realizată utilizînd limbajul **APS (Advanced Programming Software)** pentru controller-ele programabile din gama Allen Bradley şi **Wonderware Intouch** pentru proiectarea interfeței utilizator; principiul de bază l-a reprezentat minimizarea pe cît posibil a programului de control din PLC (şi implicit obținerea unui timp de scanare mai mic) prin tranferul tuturor procesărilor complementare către PC (paragraful 6.4.)

De asemenea, trebuie precizat faptul că testarea a fost realizată într-o manieră dinamică, de-a lungul tuturor fazelor de dezvoltare a sistemului, conform schemei de **testare dinamică** prezentată în Figura 8. Astfel, în fazele inițiale, testarea s-a realizat utilizînd modelul simulat al procesului de controlat; abia în ultima fază (cea de implementare) testarea a fost realizată în timp real, adică direct asupra procesului real.

Din acest motiv realizarea simulării (modelul $\mathbf{M_p}$) a reprezentat un element cheie în cadrul procesului de dezvoltare a aplicației în ansamblul său, astfel încît un efort considerabil a fost depus pentru a crea un model simulat corect (verificat şi validat). **Testarea interactivă** a reprezentat o unealtă deosebit de utilă atît în realizarea şi validarea atît a acestui model cît şi ulterior, în procesul de testare în timp real a aplicației implementate.

De asemenea, trebuie precizat faptul că aplicarea metodologiei din cadrul capitolului 5 n-ar fi fost posibilă fără existența unor unelte CASE şi medii software adecvate. Combinația utilizată în acest caz, şi anume:

- ACSL/Graphic Modeller
- Wonderware InTouch
- APS

reprezintă o variantă propusă şi care a fost utilizată în acest caz concret, nicidecum o obligativitate. Ea ilustrează însă modul în care astfel de unelte software pot fi combinate în scopul dezvoltării de aplicații software timp real.

Astfel, aplicația de monitorizare şi control a instalației geotermale de la Universitatea din Oradea a arătat că, pentru o largă categorie de sisteme timp real şi anume ***sisteme de control automat a proceselor industriale***, metodologia propusă constituie o soluție de dezvoltare simplă şi fezabilă din punct de vedere practic, utilizarea paradigmei ciclice bazate de asigurarea resurselor necesare (resource adequacy) făcînd posibilă asigurarea cerințelor de predictabilitate şi determinism impuse aplicațiilor timp real.

7. CONCLUZII FINALE ŞI CONTRIBUŢII ORIGINALE

După cum a fost deja precizat în cadrul capitolului 1, un sistem timp real este un sistem care controlează un mediu prin recepționarea datelor, prelucrarea lor şi efectuarea unor acțiuni sau furnizarea unor rezultate suficient de repede pentru a-i controla evoluția. In esență, aceasta presupune ca rezultatele să se obțină în timp util pentru a putea avea un efect nemijlocit asupra sistemului, deci să nu aibă doar o valoare pur informativă, post factum, ca şi în cazul tradițional.

Generalitatea definiției anterioare, conferă sistemelor timp real o mare diversitate, ele putînd practic acoperi domenii dintre cele mai variate, de la o navă cosmică pînă la un student care se instruieşte asistat de un calculator. Totuşi, datorită ponderii însemnate pe care se preconizează că o vor deține îndeosebi în Romania în următorii ani, s-a avut în vedere în cadrul acestei teze o categorie specifică de sisteme timp real, şi anume ***sisteme timp real utilizate pentru conducerea şi controlul proceselor industriale***. Din acest punct de vedere teza aduce în discuție un domeniu de mare actualitate.

Astfel, principalele caracteristici ale unui sistem timp real utilizat pentru controlul proceselor industriale pot fi sintetizate prin următoarele:

- existența unei mulțimi de activități care se desfăşoară în acelaşi timp şi sunt intens dependente între ele
- activitățile paralele supuse controlului depind de o multitudine de factori externi; sistemul trebuie să reacționeze prompt, flexibil şi adecvat împrejurărilor
- comenzile trebuie realizate într-un timp limitat superior în mod net, depăşirea limitei concretizîndu-se într-o situație de alarmă şi care trebuie tratată în consecință
- întrucît mărimile din cadrul unui proces industrial pot fi foarte diferite în natura lor, măsurarea lor şi execuția comenzilor presupune utilizarea unui echipament diversificat şi complex de interfațare cu procesul industrial
- funcționarea procesului este în general continuă, chiar dacă procesul în sine este discret; rolul factorului uman este într-un anumit număr de situații esențial, el trebuind inclus în buclele de reglare

Avînd în vedere aceste caracteristicile de mai sus, necesitatea elaborării unor metode de proiectare şi tehnici de implementare specifice acestei categorii de aplicații timp real este evidentă.

7.1. Contribuții teoretice

In contextul realizării de aplicații timp real pentru controlul proceselor industriale, principala ***contribuție teoretică*** pe care o aduce teza o reprezintă ***definirea unei metodologii de proiectare noi şi originale*** specifice aplicațiilor timp real pentru controlul proceselor industriale, prin aplicarea căreia să se asigure cerințele temporale, de predictabilitate şi dependabilitate specifice acestei categorii de sisteme.

Fiind orientată către aplicații care au o puternică componentă de timp real: control, achiziție şi prelucrare masivă de date, metodologia combină în esență principii şi tehnici cunoscute din cadrul ingineriei programării IP şi a ingineriei controlului automat ICA cu ***paradigme şi abordări originale***, furnizînd pe ansamblu un set de ***concepte***, ***criterii*** şi ***etape*** împreună cu categoriile de ***unelte software*** necesare implementării practice a metodologiei.

Astfel, în construirea metodologiei propuse s-a pornit de la ideea că principalul element care trebuie considerat în procesul de proiectare al unui sistem de control timp real îl reprezintă posibilitatea ***garantării*** satisfacerii tuturor constrîngerilor de timp impuse; acest obiectiv a putut fi realizat prin alegerea in primul rînd al ***simplității*** ca principiu fundamental al proiectării.

Pentru a putea realiza acest lucru, la baza modelării sistemului s-a aflat ***paradigma de răspuns garantat***, ceea ce, după cum a fost precizat în cadrul capitolului 2, implică automat asigurarea tuturor resurselor necesare, ***resource adequacy***. In plus, simplitatea modelului a reprezentat un element esențial în construirea acestuia, în măsura în care reducerea numărului de informații sau abstractizarea în vederea simplificării nu a afectat corectitudinea acestuia.

Chiar dacă utilizarea tehnologiei obiectuale este extrem de răspîndită în prezent în aplicațiile care nu sunt de timp real, metodologia propusă a fost dezvoltată avînd la bază tehnici non-obiectuale. S-a preferat această abordare datorită faptului că, după cum s-a arătat în cadrul capitolului 3, pînă în momentul elaborării acestei lucrări tehnologia orientată pe obiecte nu se afla încă în stadiul de a fi utilă pentru aplicațiile timp real complexe, existînd necesitatea îmbunătățirii metodologiilor obiectuale existente atît din punctul de vedere al semanticii temporale, cît şi a specificării, customizării, catalogării şi regăsirii componentelor software. Chiar dacă actualmente există numeroase unelte CASE disponibile, acestea nu sunt capabile să rezolve problema sistemelor în timp real şi distribuite de dimensiuni mari: de exemplu, simulatorul **ObjectGeode** reprezintă o facilitate importantă însă, în cazul aplicațiilor în timp real complexe, utilizarea modului de simulare exhaustivă poate deveni practic imposibilă datorită gradului mare de complexitate a acesteia.

In scopul atingerii dezideratelor de predictabilitate şi determinism, metodologia propusă se bazează pe utilizarea ***paradigmei Cy-Clone*** **(Clone of the Cyclic Paradigm)** a cărei idee de bază este aceea aceea că există suficiente resurse în sistem astfel încît să se garanteze că toate operaţiile vor fi executate în timp util, ***în combinaţie cu PLC-urile***. Datorită modului de operare caracteristic PLC-urilor, s-a demonstrat în cadrul capitolului 4, că acestea se pretează foarte bine la utilizarea în sistemele de control timp real time-triggered **TT**, şi care sunt proiectate utilizînd paradigma Cy-Clone.

Astfel, utilizarea controlului temporal **TT**, specific paradigmei ciclice, precum şi al **PLC**-urilor în procesul de implementare reduce semnificativ complexitatea aplicaţiei prin realizarea unei discipline temporale stricte. Prin utilizarea unei astfel de abordări, este posibilă atingerea dezideratelor de predictabilitate şi determinism în sistemele de control timp real: aceasta însemnînd în esenţă că, dîndu-se un set de cerinţe, se poate garanta că sistemul le îndeplineşte întotdeauna.

Metodologia constă în principal din următoarele etape, şi care au fost prezentate în detaliu în cadrul capitolului 5:

- realizarea modelului descriptiv $\mathbf{M_d}$ pe baza specificaţiilor iniţiale ale sistemului; modelul $\mathbf{M_d}$ include şi strategia de control adoptată
- realizarea modelului prescriptiv $\mathbf{M_p}$ şi validarea acestuia utilizînd simularea
- realizarea modelului de implementare $\mathbf{M_{impl}}$ pe baza modelului $\mathbf{M_p}$ în prealabil validat
- implementarea software şi hardware a sistemului pornind de la modelul de implementare $\mathbf{M_{impl}}$ şi testarea acestuia

In scopul aplicării în practică a metodologiei, a fost necesară utilizarea următoarelor categorii de unelte CASE şi medii de dezvoltare:

- unelte CASE pentru reprezentarea matematică **MMS** a componentelor sistemului în cadrul modelelor $\mathbf{M_d}$ şi $\mathbf{M_p}$
- un limbaj de simulare general pentru implementarea şi ecuaţiilor pentru modelul matematic $\mathbf{M_p}$ descris mai sus cu posibilitatea simulării acestuia
- un software de realizare a interfeţei utilizator care să permită managerizarea în timp real a execuţiilor precum şi crearea unei interfeţe utilizator interactive
- un software pentru implementarea programului de control propriu-zis din cadrul PLC-ului

Există o mulțime de critici care au fost aduse aplicațiilor care adoptă abordarea ciclică, şi care au fost prezentate pe larg în paragraful 4.2. Cel mai important dezavantaj îl reprezintă însă faptul că, datorită limitărilor existente ale echipamentelor hardware uilizate, nu este întotdeauna posibil din punct de vedere practic să se garanteze existența tuturor resurselor necesare, şi deci, aplicarea metodologiei nu este posibilă în aceste cazuri.

Totuşi, datorită creşterii continue a performanțelor hardware împreună cu scăderea prețului acestora, acest dezavantaj poate fi depăşit în cele mai multe cazuri. Astfel, in prezent există ***o largă categorie de aplicații industriale*** pentru care soluția propusă este perfect aplicabilă din punct de vedere practic şi pentru care aceasta furnizează o bază cunoscută pentru obținerea predictabilității şi determinismului: condiții esențiale pentru sistemele timp real. Mai mult, dacă sunt proiectate corespunzător, aceste aplicații pot furniza o comportare în timp care poate fi reprodusă (determinism reproductibil).

Pentru acestea, metodologia propusă furnizează atît un ***set de concepte şi criterii*** care pot fi utilizate în procesul de dezvoltare în ansamblu a unei aplicații de control timp real cît şi o selecție a celor mai potrivite ***unelte software*** (**CASE tools**) cu ajutorul cărora dezvoltarea aplicației să se realizeze într-o manieră cît mai simplă şi pe cît posibil, automatizată.

Cîteva dintre avantajele şi dezavantajele metodologiei propuse sunt prezentate în continuare.

Avantaje

- **simplitate** – metodologia poate fi aplicată pentru sisteme cu un grad relativ ridicat de complexitate; exemplul prezentat în studiul de caz preprezentînd o dovadă în acest sens
- **posibilitatea asigurării cerințelor de predictabilitate şi determinism** – element de o importanță majoră pentru sistemele timp real, acestea sunt asigurate datorită utilizării combinate a controlului temporal şi al PLC-urilor în procesul de dezvoltare a aplicației
- **posibilitatea utilizării tehnicilor de dezvoltare existente** în domeniul **IP** şi **ICA** precum şi a diverselor unelte şi medii software existente, în cadrul diferitelor faze ale procesului de dezvoltare

Dezavantaje

- ***lipsa de flexibilitate*** – datorită utilizării paradigmei **Cy-Clone**, şi deci implicit a controlului temporal **TT**, ordinea desfaşurării tuturor acțiunilor precum şi momentele de timp asociate acestora trebuiesc definite şi cunoscute în prealabil; aceasta cerința reprezintă o limitare față de cazul utilizării controlului bazat pe evenimente **ET;**

- ***aplicabilitate pentru cazul sistemelor timp real nu foarte rapide*** – în cazul în care constrîngerile de timp impuse pentru o aplicație dată sunt mai mici decît ciclul de bază ***dT*** caracteristic abordării ciclice (paragraful 4.2.), abordarea bazată pe evenimente ***ET*** trebuie luată în considerare din considerente de performanță (cerințe de timp de ordinul milisecundelor sau zecilor de milisecunde); astfel de sisteme nu pot fi modelate utilizînd metodologia propusă.

In plus, mai trebuie precizat faptul că sistemele timp real existente în prezent, chiar dacă prezintă unele aspecte comune, ele pot diferi foarte mult în ceea ce priveste aspecte ca: dimensiunea, complexitatea, strictețea cerințelor temporale, fiabilitatea, etc.; astfel, este foarte important ca pentru fiecare sistem particular metodologia de proiectare să fie identificată şi adaptată în consecință.

7.2.Contribuții practice

In dorința de a verifica din punct de vedere practic contribuțiile originale teoretice aduse prin intermediul metodologiei propuse, autoarea prezintă un exemplu concret de utilizare în practică a acesteia. Astfel, aplicația de control timp real **"Sistem de monitorizare şi control a instalației geotermale de la Universitatea din Oradea"** prezentată în cadrul capitolului 6 a fost dezvoltată utilizînd etapele metodologiei propuse şi descrise anterior, cu adaptările corespunzătoare impuse de modelului general de dezvoltare al unei aplicații timp real.

Dintre multitudinea de unelte CASE şi software existente s-a ales pentru aplicația prezentată în studiu de caz o combinație posibilă, şi anume: mediul **ACSL /GM (Advanced Continuous Simulation Language / Graphic Modeller)** pentru reprezentarea a componentelor sistemului şi simularea modelului generat; software-ul de **InTouch WindowsViewer** pentru crearea interfeței utilizator interactive; software-ul **Allen Bradley APS (Advanced Programming Software)** pentru implementarea programului de control din PLC. Utilizarea simulării în combinație cu testarea interactivă a permis evaluarea sistemului de control real implementat din toate punctele de vedere: funcțional, al performanțelor (şi în mod deosebit a celor temporale) precum şi al modului în care acesta reacționează în situațiile de defecțiune şi eroare, etc.

Procesul de dezvoltare a aplicației a constat în consecință din următoarele etape:

- ***Etapa 1: analiza şi specificarea cerințelor*** – au fost sistematizate cerințele sistemului în ansamblul său rezultînd un document in format informal (textual)

- ***Etapa 2: controlul algoritmic*** – au fost dezvoltate modelele $\mathbf{M_d}$ (semi-formal) şi respectiv $\mathbf{M_p}$ (formal) pentru sistemul în cauză, utilizînd mediul integrat **ACSL/Graphic Modeler;** de asemenea, în scopul realizării testării interactive a modelului $\mathbf{M_p}$ (simulării), a fost dezvoltată în paralel şi o interfață grafică cu utilizatorul utilizînd software-ul **Wonderware inTouch**
- ***Etapa 3: proiectarea software/hardware*** – a fost definitivată structura generală software/hardware a sistemului pe baza evaluării rezultatelor obținute din testarea interactivă a simulării
- ***Etapa 4: implementarea software-ului şi testarea*** – a fost implementat şi testat software-ului de control propriu-zis din cadrul PLC-ului utilizînd **Allen Bradley APS (Advanced Programming Software)**

Aplicația este funcțională din anul 1997, chiar dacă unele modificări au mai fost realizate de atunci, fără a opri însă sistemul. Prin intermediul ei se realizează optimizarea consumului de apă geotermală în funcție de cerințe. După cum a fost deja precizat în cadrul capitolul 6, apa geotermală se utilizează pentru încălzire cît şi pentru furnizarea de apă caldă menajeră în clădirile campusului. Datorită variațiilor mari ale consumului de-a lungul diferitelor perioade ale anului (ex: iarna, în timpul semestrului scolar: consum mare; vara, în timpul vacanței: consum mic), în urma implementării acestui sistem s-a realizat o importantă economie de apă.

De asemenea, alte avantaje care rezultă în urma utilizării sistemului de monitorizare şi control automat sunt:

- asigurarea unei funcționări mult mai sigure
- realizarea monitorizării continue în timp a stării procesului, ceea ce conduce la reacții foarte rapide la eventuale situații de alarmă
- posibilitatea utilizării unui minim de forță de muncă.

In concluzie, aplicația de monitorizare şi control a instalației geotermale de la Universitatea din Oradea a arătat că, pentru o largă categorie de sisteme timp real şi în particular pentru ***sisteme de control automat a proceselor industriale***, metodologia propusă constituie o soluție de dezvoltare simplă şi fezabilă din punct de vedere practic, utilizarea paradigmei ciclice bazate de asigurarea resurselor necesare (resource adequacy) în combinație cu PLC-urile făcînd posibilă asigurarea cerințelor de predictabilitate şi determinism impuse acestor categorii de aplicații.

7.3. Dezvoltări ulterioare

Actuala teză reprezintă o etapă într-un domeniu mai vast, ce constituie de mai mulți ani o preocupare a colectivului de cercetare Geotermal-Exploatare de la Universitatea din Oradea, din cadrul căruia face şi autoarea parte. Cercetările au început încă din anul 1995, continuînd în prezent cu partea de proiectare şi implementare a sistemului de monitorizare şi control a centralei electrice (Anexa 1), subsistem care nu a fost încă finalizat, acesta punînd probleme mai deosebite datorită exsitenței unor constîngeri (inclusiv temporale) mult mai severe.

In prezent se studiază posibilitatea utilizării (cu eventualele adaptări) a metodologiei propuse pentru dezvoltarea acestei aplicații de control timp real, etapa care se finalizează prin această teză avînd rolul de a oferi un prim suport teoretic şi practic pentru rezolvarea aplicațiilor de această natură.

Teza deschide de asemenea drumul către alte direcții de îmbunătățire a metodologiei propuse, eventual prin luarea în considerare şi a tehnologiilor obiectuale, odată cu îmbunătățirea semanticii temporale a acestora.

BIBLIOGRAFIE

[ACZ93] Agrawal G., Chen B., Zhao W.: *Local Synchronous Capacity Allocation Schemes for Guaranteeing Message Deadlines with the Timed Token Protocol* , INFOCOM '93, IEEE, Piscataway, NJ, 1993

[AMR98] Adan J.M., Magalhaes M.F., Ramamritham K: *Developing Predictable and Flexible Distributed Real-time systems,* Control Engineering Practice, Vol 6, 1998

[AW88] Anstrom K. J., Wittenmark B. : *Computer Controlled Systems. Theory and design*, Prentice Hall Inc., New York, 1988.

[Bar88] Barney G. *: Intelligent Instrumentation: Microprocessor Applications in Measurement and Control*, Prentice Hall, 1988

[BB99] Bate I., Burns A.: *A Framework for Scheduling in Safety-Critical Embedded Control Systems*, Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications, 1999

[Ben94] Bennett Stuart: *Real Time Computer Control*, Prentice Hall, 1994

[Beq98] Bequette B.W.: *Process Dynamics: Modeling, Analysis and Simulation*. Prentice Hall New Jersey, 1998

[Ber92] Berard Software Engineering, Inc: *A Comparison of Object-Oriented Development Methodologies*, Gaitherburg, MD: Berard Software Engineering Inc., 1992

[BF97] Barbat B., Filip F. Gh. : *Ingineria programării în timp real*, Editura Tehnică Bucureşti, 1997

[BGM95] Bonfatti F., Gadda G., Monari P.D: *Reusable software design for programmable logic controllers*, ACM SIGPLAN Workshop Proceedings, 1995

[Boa93] Boasson M.: *Control Systems Software,* IEEE Transactions on Automatic Control, vol. 38, no. 7, 1993

[Boo98] Booch G.: *Coming of Age in an Object Oriented World*, IEEE Software, November 1998

[Boy93] Boyer Stuart : *SCADA Supervisory Control And Data Acquisition*, Instrument Society of America, 1993

[BPSW99] Burns A., Punnekkat S., Stringini L., Wright D.R: *Probabilistic Scheduling Guarantees for Fault-Tolerant Real-Time Systems*, Proceedings of the 7th International Working Conference on Dependable Computing for Critical Applications, 6-8 January 1999

[Bra96] Brandin B.A.: *The Real-Time Supervisory Control of an Experimental Manufacturing Cell*, IEEE Transactions on Robotics and Automation, vol. 12, no. 1, 1996

[BRI98a] Booch G. , Rumbaugh J. , Iacobson I. *: Unified Modeling Language Reference Guide*, Adison Wesley Inc., 1998

[BRI98b] Booch G. , Rumbaugh J. , Iacobson I.: *Unified Modeling Language User Guide*, Adison Wesley Inc., 1998

[Bru95] Bruno G. : *Model-Based Software Engineering*, Chapman-Hall, London, 1995.

[BSV93] Balarin F., Sangiovanni-Vincentelli A.*: A Verification Strategy for Timing Constrained Systems,* Proceedings of Fourth International Connference on Computer Aided Verification, Springer-Verlag, 1993.

[BW91] Burns A., Wellings A. : *Real Time Systems and Their Programming Languages*, Adison Wesley Publishing , 1991

[CHLR93] Clements P.C., Heitmeyer C.L., Labaw B.G, Rose M.T. : *MT: A Toolset for Specifying and Analyzing Real Time Systems*, Proceedings Real Time Symposium, 1993

[Consi93] Considine D.M. : *Process Industrial Instruments and Controls Handbook*, 4th edition, McGraw-Hill, 1993

[Const93] Constantine L.: *Work Organization: Paradigms for Project Management and Organization* , Communication of the ACM 36, no. 10, October 1993

[Coo97] Cooling J. E. : *Real-Time Software Systems: An Introduction to Structured and Object Oriented Design*, PWS Publishing Company, 1997

[Cri90] Crispin Alan: *Programmable Logic Controllers and Their Engineering Applications*, McGraw Hill, 1990

[CVH98] Control Valve Handbook, 3rd edition, Fisher Controls International Inc., USA, 1998

[DB86] Davidoviciu A., Bărbat B.: *Limbaje de programare pentru sisteme în timp real*, Editura Tehnică Bucureşti, 1986

[Deu88] Deutsch M.S. : *Focusing Real-Time Systems Analysis on User Operations*., IEEE Software, September 1988

[Dia84] Diaz M.: *DROL: A Distributed and Real-Tine Object Oriented Logic Environment,* The Computer Journal 37, no.5, 1984

[DLZSF98] Deng Z., Liu J.W.S., Zhang L., Seri M., and Frei A.: *An Open Environment for Real-Time Applications*, Real-Time Systems Journal, 1998

[Dor92] Dorf R. C.: *Modern Control Systems*, Addison Wesley Publishing Company, 1992

[Dou98a] Douglas B. P.*: Doing Hard Time: Objects and Patterns in Real-Time Software Development*, Reading MA: Adison Wesley Longman, 1998

[Dou98b] Douglas B. P.*: Real-Time UML: Developing Efficient Objects for Embedded Systems*, Adison Wesley Inc., 1998

[Eli95] Eliens A.: *Principles of Object-Oriented Software Development*, Addison-Wesley Inc., 1995

[Ell94] Ellis J. R. : *Objectifying Real-Time Systems*, SIGS Books, New York 1994

[ESA93] ESA*: HOORA: Hierarchical Object-Oriented Requirements Analysis,* ESA Report E2S/OORA/WP1/METH, 1993

[FB95] Frensel, G., Bruijn, P.M.: *Issues in Development Models of Real-Time Process Control Software*, Preprints of the 3rd IFAC/IFIP Workshop on Algorithms and Architectures for Real-Time Control, Ostende, Belgium, 1995

[FCL93] Fruehauf P.S., Chien I.L., Lauritsen M.D.: *Simplified IMC-PID Tuning Rules* , ISA Paper # 93-414, 1993

[GF86] Guran M., Filip F.G.: *Sisteme ierarhizate şi în timp real cu prelucrarea distribuită a datelor*, Editura Tehnică Bucureşti, 1986

[GHJV94] Gamma E., Helm R., Johnson R., Vlissides J. : *Design Patterns – Elements of Reusable Object Oriented Software*, Reading, MA: Adison Wesley, 1994

[GJM91] Ghezzi C., Jazayeri M., Mandrioli D. : *Fundamentals of Software Engineering*, Prentice Hall, 1991

[Gol93] Goldsmith S.: *A practical Guide to Real-Time Systems development,* Prentice Hall, Englewood Cliffs NJ,1993

[Gom86] Gomaa H. : *Software Development of Real-Time Systems*, Communications of the ACM, vol. 29, no. 7, 1986.

[Gom89] Gomaa, H.: *A Software Design Method for Distributed Real-Time Applications*, Journal of System and Software, 1989

[Gom93] Gomaa H.: *Software Design Methods for Concurrent and Real-Time Systems*, Reading MA: Addison Wesley,1993

[Gom95] Gomaa H..: *Reusable Software Requirements and Architectures for Families of Systems*, The Journal of Systems and Software, vol. 28, no.3,1995

[GS93] Garlan D., Shaw M.: *An Introduction to Software Architecture: Advanced in Software Engineering and Knowledge Engineering*, vol.1, World Scientific Publishing, Singapore 1993

[GW93] Gilbert J. W., Wilhem R. G.: *A Concurrent Object Model for an Industrial Process-Control Application*, Journal of Object Oriented Programming, November/December 1993

[GW93] Gilbert J.W., Wilhem R.G.: *A Concurrent Object Model for an Industrial Process-Control Application*, Journal of Object-Oriented Programming 6., no.7, 1993

[Hal90] Hall A.: *Seven Myths of Formal Methods*, IEEE Software, September 1990

[HaL92] Halang W.A. : *Real Time Systems: Another Perspective*, The Journal of Systems and Software, April 1992

[HFL95] Hull D., Feng W., Liu J.W.S.: *Enhancing the Performance and Dependability of Hard Real-Time Systems,* IEEE Computer Performance and Dependability Symposium, Erlangen, Germany, April 1995

[HG93] Horning J.J., Guttag J.V.*: Larch: Languages and Tools for Formal Specification,* Springer-Verlag, 1993

[Hin92] Hinchey M.G. : *JSD, RPT & the Design of Real Time Systems*, M.Sc. disertation, Oxford University, Programming Research Group, 1992

[HM96] Heitmeier C., Mandrioli D. : *Formal Methods for Real-Time Computing*, John Wiley & Sons, 1996

[Hod98] Hodson W.R.: *A Fieldbus Primer*, Oil Patch Magazine, November/December 1998

[Hoh96] Hohmann, T.: *Why PCs Won't Kill PLCs*, *Industrial Computing, vol. 15, no. 10,*1996

[How93] Howell S. : *Annotation for System Engineering of Large Complex Real-Time Systems*, NATO Real-Time Systems Conference Proceedings, 1993

[HP87] Hartley D., Pirbhai I. : *Strategies for Real-Time System Specification*., New York: Dorset House Publishing, 1987

[HS91] Halang W. A., Stoyenko A. D. : *Constructing Predictable Real-Time Systems*, Kluwer Academic Publishers, 1991

[IBR98] Iacobson I., Booch G., Rumbaugh J.*: The Objectory Software Development Process*, Adison Wesley Inc., 1998

[IEC95] IEC Control Valve Standard, IEC 534-8-3,1995

[IEEE93] IEEE: *IEEE Guide to Software Requirements Specifications,* IEEE Std 830-1993, The Institute of Electrical and Electronics Engineers, 1993

[Irw96] Irwin G.W.: *The Engineering of Complex Real Time Computer Control*, Kluwer Academic Publisher, 1996

[ITM90] Ishikawa Y., Tokuda H., Mercer C. : *Object-Oriented Real-Time Languages Design: Constructs for Timing Constrains*, Proceedings of OOPSLA, 1990

[Jah93] Jahanian F.: *Verifying Properties of Modechart Specifications*, Proceedings Real Time Symposium, 1993

[JCJG92] Jacobson I., Christerson M., Jonsson P., Gunnar O.: *Object Oriented Software Engineering – A Use Case Driven Approach*, Reading MA: Addison Wesley, 1992

[JM94] Jahanian F., Mok A.K.: *A Specification Language for Real-Time Systems*, IEEE Transaction on Software Engineering vol. 20, 1994

[Kav92] Kavi K. M.: *Real-Time Systems - Abstraction, Languages and Design Methodologies*, IEEE Computer Society Press, 1992

[Kle93] Klein M., Ralya T., Pollak B., Obenza R., Gonzalez M.: *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate-monotonic Analysis for Real-Time Systems,* Boston, Kluwer Academic Publishers,1993

[Kop91] Kopetz H. : *Event-Triggered versus Time-Triggered Real-Time Systems*, International Workshop on Operating Systems in the 90' and Beyond, 1991

[Kop92] Kopetz H.: *The Time Triggered Approach to real Time System design - Predictability Dependable Computing Systems*, Springer Verlag, Heildelberg, 1992

[Kop97] Kopetz, H.: *Real-Time Systems - Design Principles for Distributed and Embedded Applications*, Kluwer Academic Publishers, 1997

[Kop98] Kopetz H.: *The Time-Triggered Model of Computation*, The 19th IEEE Systems Symposium (RTSS98), Madrid, Spain, 1998

[Koy90] Koymans R.: *Specifying Real Time Properties with Metric Temporal Logic*, Real-Time Systems Journal, vol. 2, no. 4, 1990

[Kru92] Krueger C. W. : *Software Reuse*, ACM Computing Surveys 24, no. 2, 1992

[KS92] Klingerman E., Stoyenko A. D. *: Real-Time Euclid: A Language for Reliable Real-Time Systems*, IEEE Transactions on Software Engineering, vol. 12, no. 9, 1992

[KS97] Krishna, C. M., Shin K. G.: *Real-Time Systems*, McGraw-Hill Companies Inc., 1997

[KTK91] Kurose J. F., Towsley D. F., Krishna C. M.*:Design and Analysis of Processor Scheduling Policies for Real-Time Systems*, Foundations of Real Time Computing: Scheduling and Resource Management, Kluwer, Boston, 1991

[KY92] Kavi K.M., Yang S.M. : *Real-Time Systems Design Methodologies: An Introduction and a Survey*, The Journal of Systems and Software, April 1992

[KY95] Kavi, K. M., Yang, S. M.: *Real-Time Systems Design Methodologies: An Introduction and a Survey*, Journal of Systems and Software, 1995

[LA93] Levi S. T., Agrawala A. K. : *Real Time Systems Design*, McGraw Hill Publishing, 1993

[Lap93] Laplante P.A.: *Real-Time Systems Design and Analysis*, IEEE Press, 1993

[Lau93] Lauber R.: *Statement on a Concept for Dynamic Testing in Four Sequential Steps During Software Development*, NATO Real-Time Systems Conference Proceedings, 1993

[Law92] Lawson H. W. : *Cy-Clone: An Approach to the Engineering of Resource Adequate Cyclic Real-Time Systems*, Journal of Real-Time Systems, Vol. 4, No,1, 1992

[LBGG94] Lee I., Bremont-Gregoire P., Gerber R.: *A Process Algebraic Approach to the Specification and Analysis of Resource-Bound Real-Time Systems*, Proceedings of IEEE, January 1994

[Lev95] Levine P. S.: *The Programmable (Logic) Controller: Adapting in an Environment of Change, Industrial Computing, vol. 14, no. 3*, 1995

[LH91] Lala J. H., Harper R. E. *:A Design Approach for Ultra-Reliable Real-Time Systems*, IEEE Computer no. 24, 1991

[LH95] Liu J.W.S., Ha R.: *Methods for Validating Real-Time Constraints,* The Journal of Systems and Software, vol. 30 (1-2), July-August 1995

[LL73] Liu C.L., Layland J. W.: *Scheduling Algorithm for Multiprogramming in a Hard Real Time Environment* , Journal of the ACM, vol. 20, no. 1, 1973

[LRD95] Lagnier F., Raymond P., Dubois C. : *Formal verification of critical systems written in Saga/Lustre*, Workshop of Formal Methods, Modelling and Simulationfor System Engineering, St. Quentin en Yvelines, France, February 1995

[LSGL94] Luchangco V., Soylemez E., Garland S., Lynch N. : *Verifying Timing Properties of concurrent Algorithms*, Proceedings of Formal Description Techniques VII FORTE, 1994

[LSLBC91] Liu J. W. S., Shin W. K., Lin K. J., Bettati R., Chung J. Y. *Algorithms for Scheduling Imprecise Computations*, Foundations of Real Time Computing: Scheduling and Resource Management, Kluwer, Boston, 1991

[Mar95] Marlin T.E.:*Process Control: Designing Processes and Control Systems for Dynamic Performance,* McGraw Hill, New York, 1995

[McD93] McDermid J.: *Dependability Requirements: Orthodoxy and Goal-structured Approach* , Proceedings of the Dependable Software Technology Exchange, Software Engineering Institute, Carnergie Mellon University, March 1993

[MGA95] MGA Software: *Advanced Continuous Simulation Language ACSL. Reference Manual*, 1995

[MGA96a] MGA Software: *Graphic Modeller. User's Guide*, 1996

[MGA96b] MGA Software: *Advanced Continuous Simulation Language ACSL. User's Guide*, 1996

[Mic90] Michel G. *: Programmable Logic Controllers*, Wiley & Sons, New York, 1990

[MK94] Ma J., Knight B.: *A General Temporal Theory*, The Computer Journal 37, no.2, 1994

[OGF84] Orăşanu L., Gaşpar F., Filip F.G.: *Two CAD Systems of Real-Time Control Systems for Complex Systems*, Real Time Hierarchical Control, Springer-Verlag, 1984

[OR94] Ogunaike.A., Ray W.H.:*Process Dynamics, Modeling and Control*, Oxford University Pressinc., New York, 1994

[Ost92] Ostroff J.S. : *Formal Methods for the Specification and Design of Real Time Safety Critical Systems*, The Journal of Systems and Software, April 1992

[Ost94] Ostroff J. : *Visual Tools for Verifying Real-Time Systems*, Theories and Experiences for Real-Time System Development, AMAST Series in Computing, vol. 2, Singapore 1994

[OW90] Ostroff J.S., Wonham W.M.: *A Framework for Real-Time Discrete Event Control*, IEEE Transactions on Automatic Control, vol 35, no. 4, 1990

[Pag88] Page-Jones M. : *Practical Guide to Structured Systems Design*., 2nd edition, Prentice Hall, Englewood Cliffs NJ, 1988

[Par95] Parr E. A. : *Programmable Controllers. An engineer's Guide*, Newnes, an imprint of Butterworth-Heinemann, 1995

[Per91] Perdikaris G. A.: *Computer Controlled Systems. Theory and Applications*, Kluwer Academic Publisher, 1991

[Pes93] Pesonen P.: *Object-Based Design of Embedded Software*, Technical Research Center of Finland, 1993

[Pfl91] Pfleeger S. L. *: Software Engineering*, 2nd edition, New York, Macmillan, 1991

[PH96] Philips C. I., Harbor R. D. : *Feedback Controlled Systems*, Prentice Hall International., New York, 1996

[PM98] Puchol C., Mok A.K.*: Integrated Design Tools for Hard Real-time Systems*, The 19th IEEE Systems Symposium (RTSS98), Madrid, Spain, 1998

[Pop86] Popa B. : *Manualul inginerului termotehnician.*, Editura Tehnică, Bucureşti., 1986.

[Ram96] Ramamritham K : *Predictability: Demonstrating Timing Requirements*, ACM Computing Surveys, 28A(4), December 1996

[Rat98] Rational Software Inc.: *Real-Time Rational Rose – User Guide*, 1998

[RD89] Rodd M.C.., Deravi F. : *Communication Systems for Industrial Control*, Prentice Hall, 1989

[Rhe97] Rheinhart N. F.: *Impact of Control Loop Performance on Process Profitability*, Aspen World'97, Boston, Massachusetts, USA, October 15, 1997

[Rip89] Ripps L.: *An Implementation Guide to Real-Time Programming*, Prentice Hall, 1989

[RIZ90] Rodd M. G., Izikowitz A., Zhao G. F. : *RTMMS - An OSI-based Real-Time Messaging System*, International Journal of Real-Time Systems, vol. 2, 1990

[Roc94a] Rockwell Automation : *SLC Family of Small Programmable Controllers. System Overview*, 1994

[Roc94b] Rockwell Automation : *Advanced Programming Software (APS). User manual*, 1994

[Roc94c] Rockwell Automation : *Advanced Programming Software (APS). Reference manual*, 1994

[Roc95] Rockwell Automation : *Allen Bradley Automation Systems*, 1995

[Rod93] Rodd M.C.: *Communication for Real-Time Industrial Control: The Design Issues*, NATO Real-Time Systems Conference Proceedings, 1993

[RSS90] Ramamritham K. J., Stankovic A., Shiah P. F. : *Efficient Scheduling Algorithms for Real Time Multiprocessor Systems*, IEEE Transactions on Parallel and Distributed Systems, vol 1, 1990

[Rul97] Rullan, A.: *Programmable Logic Controllers versus Personal Computers for Process Control*, Computers industrial Engineering, vol 33, no.1-2, 1997

[Sad89] Saden B. : *Entity Life Modeling and Structured Analysis in Real-Time Software Design - A Comparison*, Communication of the ACM, vol 32, no. 12, 1989

[SC85] Smith C.A., Corripio A.B.: *Principles and Practice of Automatic Process Control*, John Wiley & Sons, New York, 1985.

[SC97] Smith C.A., Corripio A.B.: *Principles and Practice of Automatic Process Control*, 2nd edition, John Wiley & Sons, New York, 1997

[SEM89] Seborg D.E., Edgar T.F., Mellichamp D.A.:*Process Dynamics and Control*, John Wiley and Sons, New York, 1989

[SGW94] Selic B., Gullekson G., Ward P. :*Real-Time Object Oriented Modeling*, John Wiley & Sons, New York, 1994

[SHH91] Stoyenko A.D., Hamacher V.C., Holt R.C.: *Analyzing Hard Real-Time Programs for Guaranteed Schedulability* , IEEE Transactions of Software Engineering, vol. 17, no. 8, 1991

[Shi96] Shinskey F.G. : *Process Control Systems: Application, Design and Tuning*, 4th edition, McGraw Hill, New York, 1996

[SMY00] Svrcek W., Mahoney D., Young B.: *A Real-Time Approach to Process Control*, John Wiley & Sons, 2000

[SR90] Stankovic I., Ramamrithan K. : *What is Predictability for Real-Time Systems?* , Real-Time Systems no. 2, 1990

[SR93] Stankovic J., Ramamritham K.: *Advances in Hard Real-Time Systems,* IEEE Computer Society Press, September 1993

[SR98] Stankovic J., Ramamritham K : *Deadline Scheduling for Real-time systems: EDF and Related Algorithms,* Kluwer Academic Publishers, 1998

[SRS94] Sha L., Rajkumar R., Sathaye S. S. : *Generalized Rate-Monotonic Scheduling Theory: A Framework for Developing Real Time Systems*, Proc. IEEE vol 82, 1994

[SS94] Sathaye S. S., Strosnider J. K. : *Conventional and Early Token Release Scheduling Models for the IEEE 802.5 Token Ring*, Real Time Systems, no. 7, 1994

[SSDNB95] Stankovic J., Spuri M., Di Natale M., and Buttazzo G.: *Implications of Classical Scheduling Results For Real-Time Systems,* IEEE Computer, Vol. 28, No. 6, 1995

[Sta88] Stankovic J. : *Misconceptions about Real-Time Computing: A serious problem for next generation systems*, IEEE Computer Magazine, vol. 21, 1988

[Sta92a] Stankovic J.: *Distributed Real-Time Computing: The Next Generation,* special issue of Journal of the Society of Instrument and Control Engineers of Japan, Vol. 31, No. 7, 1992

[Sta92b] Stankovic J.: *Real-Time Computing*, *BYTE Magazine*, invited paper, August 1992

[Sto92] Stoyenko A.D. : *The Evolution and State-of-the-Art of Real Time Languages* , The Journal of Systems and Software, April 1992

[TBW94] Tindel K.W., Burns A., Wellings A.J. : *An Extensible Approach for Analysing Fixed Proirity Hard Real-Time Tasks*, Real Time Systems no. 6, 1994

[TCGDM95] Thoen F., Cornero M., Goossens G., De Man H.: *Software sinthezis for real-time information systems*, ACM SIGPLAN Workshop Proceedings, 1995

[TL92] Tyreus B.D., Luyben W.L.: *Tuning of PI Controllers for Integrator/Deadtime Processes* , Ind. Eng. Chem. Res., 1992

[Ver96] Verilog Inc.: *ObjectGEODE - Object Oriented Real-Time Techniques. Method Guidelines*, 1996

[VMP92] Veillette R. J., Medanic J. V., Perkins W. R.: *Design of Reliable Control Systems*, IEEE Transactions on Automatic Control, vol. 37, no.3,1992

[VTK91] Van Tilborg A. M., Koob G.M.: *Foundations of Real-Time Computing: Scheduling and Resource Management.* Kluwer Academic Publishers, Norwell, Massachusetts, 1991

[War88] Warnock I. G. : *Programmable Controllers. Operation and application.*, Prentice Hall Inc., New York, 1988

[Wil95] Wilner D.: *WindView: A Tool for Understanding Real-Time Embedded Software through System Visualization*, ACM SIGPLAN Workshop Proceedings, 1995

[WM85a] Ward P.T., Mellor S.J.: *Structured Development for Real-Time Systems*, Yourdon Press, New York, 1985.

[WM85b] Ward P.T., Mellor S.J.: *Structured Development for Real-Time Systems: Introduction and Tools*, Englewood Cliffs, NJ: Prentice Hall, 1985.

[Won90] Wondeware Corporation: *DDE Server. Allen Bradley Serial*, 1990

[Won94a] Wondeware Corporation : *InTouch. User Guide*, 1994

[Won94b] Wondeware Corporation : *NetDDE for Windows*, 1994

[Xu93] Xu J. : *Multiprocessor Scheduling of Processes with Release Times, Deadlines, Precedence and Exclusion Relations*, IEEE Transactions of Software Engineering, vol 19, 1993

[You89] Young S. J. : *Real Time Languages: Design and Development*, John Wiley & Sons, 1989

[Zma95] Zmaranda D.: *Automatic Control and Monitoring System for the District Heating System at the University of Oradea*, United Nations University Reports, 1995

[Zma95a] Zmaranda D.: *Optimizarea planificării task-urilor în sisteme de timp real*, Proceedings of EMES'95, Băile Felix, Romania, 1995

[Zma95b] Zmaranda D.: *Algoritmi de sortare adaptivi*, Proceedings of EMES'95, Băile Felix, Romania, 1995

[Zma96] Zmaranda D.: *Considerations regarding SCADA system programming used in the geothermal plant at the University of Oradea*, Proceedings of RSEE'96, Băile Felix, Romania,1996

[Zma97] Zmaranda D.: *Aspecte privind comunicarea în sistemele de control în timp real şi distribuite*, Proceedings of EMES'97, Băile Felix, Romania, 1997

[ZG98] Zmaranda D., Gabor G.: *Techniques and strategies used in industrial computer control*, Proceedings of RSEE'98, Băile Felix , Romania, 1998

[Zma98a] Zmaranda D.: *State of the art of the formal methods used for real-time systems*, International Scientific Conference on Electronic Computers and Informatics, Kosice, Slovacia, 1998

[Zma98b] Zmaranda D.: *The role of standards in real-time systems*, Proceedings of RSEE'98, Băile Felix, Romania, 1998

[Zma98c] Zmaranda D.: *Issues regarding the use of object-oriented approach in real-time programming languages*, Transactions on Automatic Control and Computer Science, Timişoara, 1998

[Zma99] Zmaranda D.: *Paradigms for constructing predictable real-time systems. The cyclic approach*, International Scientific Conference on Electronic Computers and Informatics, Kosice, Slovacia 1999

[ZC00] Zmaranda D., Cretu V.: *A methodology for development of mathematical models and simulation for real-time control applications*, Transactions on Automatic Control and Computer Science, Timişoara, 2000

[ZG00] Zmaranda D., Gabor G.: *State of the art of the automatic control and monitoring system from the geothermal plant from the University of Oradea, Romania: present and perspectives*, World Geothermal Congress 2000, Kyushu-Tohoku, Japonia, 2000

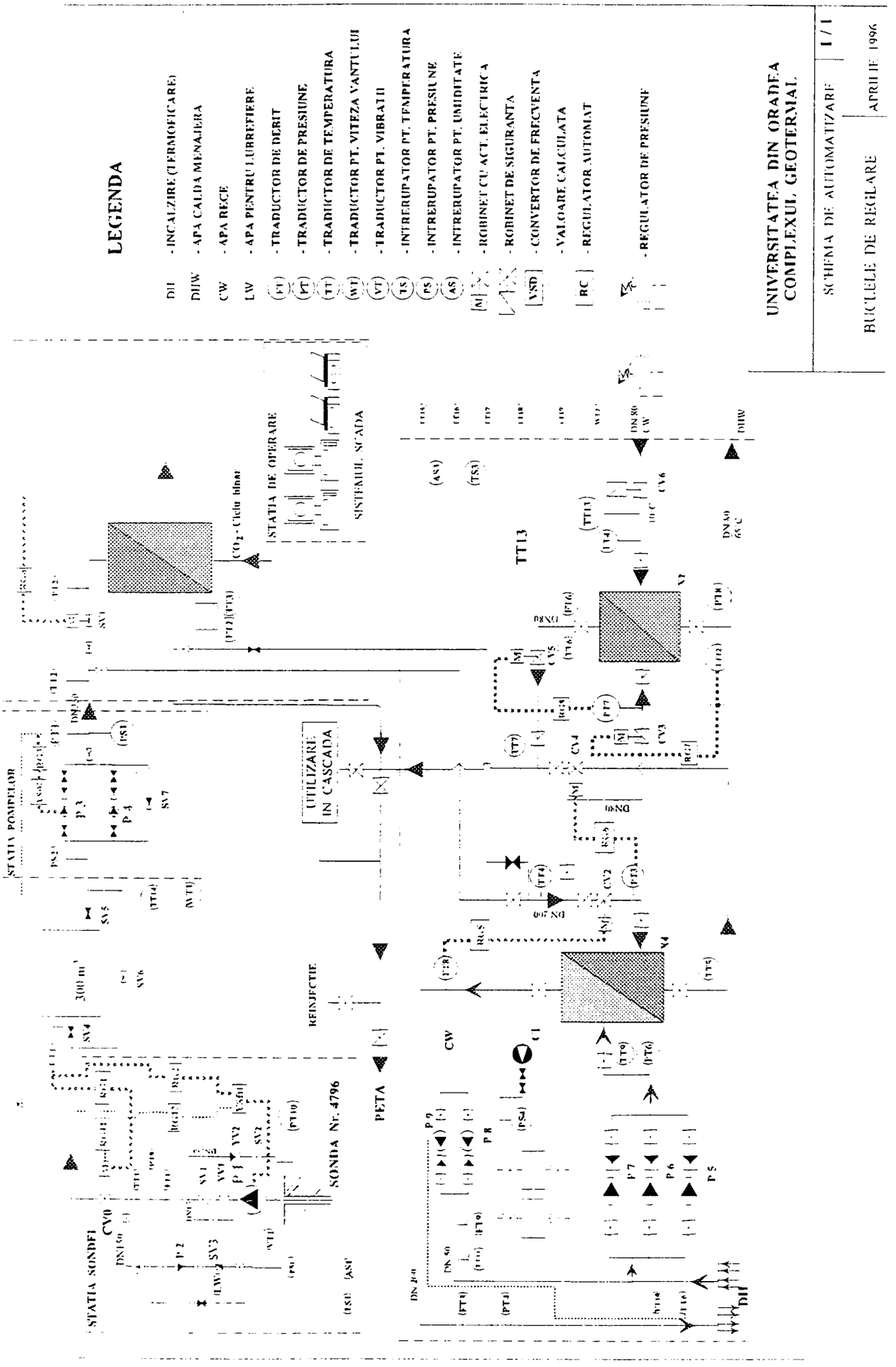
[Zma00] Zmaranda D.: *Real-time systems methodologies – a comparison. Present and perspectives*, International Scientific Conference on Electronic Computers and Informatics, Kosice, Slovacia, 2000

[ZC01] Zmaranda D., Cretu V.: *Development Issues For Real-Time Control Applications*, IEEE International Conference on Telecommunication, Bucharest, Romania, 2001

GLOSAR

ACSL	= Advanced Continuous Simulation Language
APS	= Advanced Programming Software
DDE	= Dynamic Data Exchange
ET	= Event Triggered
FSD	= Finite State Diagram
HMS	= Hierarchical Multistate Machine
ICA	= Ingineria Controlului Automat
I/O	= Input/Output
IP	= Ingineria Programării
JSC	= Jackson Structured Charts
MMS	= Modular Modeling System
MSC	= Message Sequence Charts
OMT	= Object Modeling Technique
OSI	= Open System Interconnection
PLC	= Programmable Logic Controller
RTMMS	= Real Time Manufacturing Message Specification
SCADA	= Supervisory Control And Data Acquisition
SDL	= Specification and Description Language
TT	= Timed Triggered
WCET	= Worst-Case Execution Time

Anexa 1: Schema de automatizare



Anexa 1: Valorile calculate în schema de automatizare

Notație folosită	Mod de calcul - utilizare
TT14	temperatura exterioară calculată ca valoarea medie de-a lungul unui interval de 10 minute
WT1	viteza vîntului calculată ca valoarea medie de-a lungul unui interval de 10 minute
WT2	media celor 144 valori măsurate ale lui WT1, şi calculate pentru întreaga zi (pentru 24 ore – calculată la ora 24.00 a zilei curente)
TT15	media aritmetică a celor 144 valori măsurate şi calculate ale lui TT14 pentru întreaga zi (24 ore – calculată la ora 24.00 a zilei curente).
TT16	media aritmetică a celor 144 valori măsurate şi calculate ale lui TT14 pentru ultimele 24 de ore (calculată la fiecare oră şi utilizată pentru a calcula setpoint-ul pentru regulatorul RG5).
TT17	temperatura exterioară echivalentă calculată din TT14 şi WT1 pe baza următoarei ecuații: $TT17 = TT14 - (1+0.022 \times (TT14-10.4)) \times (1.94 \times WT1)^{0.5}$
TT18	media temperaturii echivalente exterioare pe 24 ore (media celor 144 valori ale lui TT17 calculate pentru întreaga zi)
TT19	temperatura exterioară medie echivalentă pe ultimele 24 ore (media ultimelor 144 valori ale lui TT17 măsurate şi calculate). Valoarea lui TT19 poate fi folosită pentru a calcula valoarea (prestabilită) a setpoint-ului pentru regulatorul RG5, în loc de a folosi TT16, dacă se dovedeşte mai relevantă includerea influenței vântului în cererea de căldură
TT20	TT20 temperatura agentului termic furnizat în rețeaua de încălzire, reprezentînd valoarea presetată (setpoint-ul) pentru regulatorul RG5. Deoarece nu există încă instalate traductoare de căldură în clădirile universității, ceea ce ar permite reglarea temperaturii în funcție de cerere, temperatura apei furnizate poate fi reglată funcție de condițiile meteorologice prin ecuația TT8 = TT20, unde: $TT20 = x_{h14}$ dacă $TT16 > x_{h9}$ $TT20 = x_{h12} - x_{h13} \times TT16$ dacă $x_{h10} \leq TT16 \leq x_{h9}$ $TT20 = x_{h11}$ dacă $TT16 < x_{h10}$ (vezi Anexa 3)

Anexa 2: Alarme în sistem

1. Alarme la stația sondei

Alarma	Notația folosită
Presiune prea mică in puț, mai mult de 5 s	PT9_L
Presiune prea mare în puț, mai mult de 5 s	PT9_H
Presiune excesiv de mică în puț, mai mult de 5 s	PT9_LL
Presiune excesiv de mare în puț, mai mult de 5 s	PT9_HH
Vibrații mari ale pompei P1, mai mult de 5 s	VT1_H
Vibrații excesiv de mari ale pompei P1, mai mult de 5 s	VT1_HH
Debit prea mare, mai mult de 5 s	FT1_H
Debit excesiv de mare, mai mult de 2 minute	FT1_HH
Curent prea mare la motorul pompei P1, mai mult de 5 s	P1_MC_H
Curent excesiv de mare la motorul pompei P1, mai mult de 5 s	P1_MC_HH
Temperatură prea mică, mai mult de 5 s	TT1_L
Temperatură prea mare, mai mult de 5 s	TT1_H
Temperatură excesiv de mică, mai mult de 5 s	TT1_LL
Temperatură excesiv de mare, mai mult de 5 s	TT1_HH
Nivel prea mic al apei în rezervor, mai mult de 5 s	LT1_L
Nivel prea mare al apei în rezervor, mai mult de 5 s	LT1_H
LT1 mai mic decît limita LL pentru 2 minute cu ieşirea regulatorului RG2 100%	LT1_LL
LT1 mai mare decît limita HH pentru 2 minute cu ieşirea regulatorului RG2 0%	LT1_HH
Umiditate prea mare în interiorul stației sondei	AS1
Temperatură prea mare în interiorul stației sondei	TS1
Presiune prea mică a apei de ungere	PS1
Cădere de tensiune în stația sondei	PFS1

Alarme datorate defecțiunilor echipamentelor	Notația folosită
VSD1 nu este disponibil	P1_OK
VSD1 defect	P1_FA
Supraîncălzire la motorul pompei P1	P1_OH
Supraîncărcare sau scurt-circuit la motorul pompei P1	P1_OL
Pompa P1 defectă	P1_TD
Supraîncărcare sau scurt-circuit la motorul pompei P2	P2_OL
Pompa P2 defectă	P2_TD
Robinetul de reglare CV0 defect	CV0_OL
Eroare de închidere la robinetul de reglare CV0	CV0_CL
Eroare de deschidere la robinetul de reglare CV0	CV0_OP
Eroare de mişcare la robinetul de reglare CV0	CV0_M

Transmițătorul PT9 defect	PT9_TR
Transmițătorul PT10 defect	PT10_TR
Transmițătorul LT1 defect mai mult de t_{w13} secunde	LT1_TR
Transmițătorul FT1 defect	FT1_TR
Transmițătorul TT1 defect	TT1_TR
Transmițătorul VT1 defect	VT1_TR

2. Alarme la stația de pompare

Alarma	**Notația folosită**
Presiune prea mică in sistem, mai mult de 5 s	PT1_L
Presiune prea mare în sistem, mai mult de 5 s	PT1_H
Presiune excesiv de mică in sistem, mai mult de 5 s	PT1_LL
Presiune excesiv de mare în sistem, mai mult de 5 s	PT1_HH
Curent prea mare la motorul pompei P3, mai mult de 5 s	P3_MC_HH
Curent excesiv de mare la motorul pompei P3, mai mult de 5 s	P3_MC_H
Curent prea mare la motorul pompei P4, mai mult de 5 s	P4_MC_HH
Curent excesiv de mare la motorul pompei P4, mai mult de 5 s	P4_MC_H
Umiditate prea mare în interiorul stației sondei	AS2
Temperatură prea mare în interiorul stației sondei	TS2
Presiune de absorbție prea mică	PS2
Presiune de refulare prea mare	PS3
Cădere de tensiune în stația sondei	PFS2

Alarme datorate defecțiunilor echipamentelor	**Notația folosită**
VSD2 nu este disponibil	P3_OK
VSD2 defect	P3_FA
Supraîncărcare sau scurt-circuit la motorul pompei P3/P4	P3_OL /P4_OL
Pompa P3/P4 defectă	P3_TD /P4_TD
Transmițătorul PT1 defect mai mult de t_{p1} secunde	PT1_TR
Transmițătorul TT14 defect mai mult de t_{h3} secunde	TT14_TR

3. *Alarme la punctul termic*

Alarma	**Notația folosită**
Presiunea furnizată excesiv de mare pentru mai mult de t_{h6} secunde	PT4_HH$_{th6}$
Presiunea furnizată prea mare	PT4_H
Presiunea furnizată prea mică	PT4_L
Presiunea furnizată excesiv de mică	PT4_LL
Presiunea de absorbție excesiv de mică pentru mai mult de t_{h7} secunde	PT5_LL$_{th7}$
Presiunea de absorbție prea mică	PT5_L
Presiunea de absorbție prea mare	PT5_H
Presiunea de absorbție excesiv de mare	PT5_HH
Presiunea de refulare a pompei excesiv de mare pentru mai mult de t_{h8} secunde	PT6_HH$_{th8}$
Presiunea de refulare a pompei prea mare	PT6_H
Presiunea de refulare a pompei prea mică	PT6_L
Presiunea de refulare a pompei excesiv de mică	PT6_LL
Presiunea diferențială pe schimbătoarele de căldură este mai mare decît x_{h2} pentru mai mult de t_{h9} secunde	PT6_PT3
Umiditate prea mare în interiorul punctului termic	AS3
Temperatură prea mare în interiorul punctului termic	TS3
Presiune prea mică în amortizoarele pneumatice	PS4
Cădere de tensiune în punctul termic	PFS3

Alarme datorate defecțiunilor echipamentelor	**Notația folosită**
Supraîncărcare sau scurt-circuit la motorul pompelor P5-P9	P5_OL.... P9_OL
Pompele P5-P9 defecte	P5_TD.... P9_TD
Robinetele de reglare CV2 - CV6 defecte	CV2_FAULT.. CV6_FAULT
Transmițătorul PT3 defect mai mult de t_{h3} secunde	PT3_FAULT
Transmițătorul PT7 defect mai mult de t_{h3} secunde	PT7_FAULT
Transmițătoarele PT4 – PT6, PT8 defecte	PT4_FAULT.. PT8_FAULT
Transmițătorul TT12 defect mai mult de t_{h3} secunde	TT12_FAULT
Transmițătorul TT8 defect mai mult de t_{h3} secunde	TT8_FAULT
Transmițătoarele TT4 –TT7, TT9 – TT11 defecte	TT4_FAULT.. TT11_FAULT
Transmițătoarele FT3 – FT6 defecte	FT3_FAULT.. FT6_FAULT

Anexa 3: Constante utilizate în programul de control

1. *Constante pentru stația sondei*

Constante pentru evaluarea stării pompelor

c_{w1} – numarul de ore de funcționare pentru pompa de adîncime P1
c_{w2} – numarul de ore de funcționare pentru pompa de ungere P2
c_{w3} – numărul de porniri/oră a pompei P1

Constante de timp

t_{w1} – timpul de întîrziere permis PT10 ≥ (PT9-x_{w1}) înainte de schimbarea către curgere arteziană

t_{w2} – întîrzierea permisă de la oprirea pompei P1 pînă la oprirea pompei P2

t_{w3} – întîrzierea permisă de la semnalarea alarmei LT1 ≥ HH şi oprirea pompei P1 (RG2 0% ieşire)

t_{w4} – întîrzierea permisă de la semnalarea alarmei LT1 ≤ LL cu ieşirea RG2 100% - semnal transmis către centrala electrică

t_{w5} – întîrzierea permisă de la semnalarea alarmei LT1 ≤ LL şi pornirea pompei P1

t_{w6} – întîrzierea permisă după semnalarea alarmei PT9 ≥ HH şi oprirea pompei P1

t_{w7} – pasul de timp utilizat în schimbarea set-point-ului regulatorului RG2

t_{w8} – întîrzierea de la pornirea pompei P1 după care se începe schimbarea set-point-ului regulatorului RG2

t_{w9} – întîrzierea permisă de la semnalarea alarmei VT1≥ HH pînă la oprirea pompei P1

t_{w10} – întîrzierea permisă de la semnalarea alarmei PT9 ≥ H pînă la oprirea pompei P2

t_{w13} – întîrzierea permisă după detectarea unei defecțiuni la transmițătorul LT1, variabila de process pentru regulatoarele RG1 şi RG2

t_{w14} – pasul de timp utilizat în schimbarea set-point-ului regulatorului RG12

t_{w15} – întîrzierea de la pornirea pompei P1 după care se începe schimbarea set-point-ului regulatorului RG12

t_{p1} – întîrzierea permisă după detectarea unei defecțiuni la transmițătorul PT1, variabila de process pentru regulatoarele RG11 şi RG12

t_{p4} – întîrzierea permisă de la semnalarea alarmei PT1 ≥ HH şi oprirea pompei P1 (RG12 cu ieşirea 0%)

t_{L1} – întîrzierea permisă pentru închiderea robinetului CV0 – stop operare, defecțiune

t_{L2} – întîrzierea permisă pentru deschiderea robinetului CV0 – stop operare, defecțiune

t_{L3} – întîrzierea permisă pentru operarea robinetului CV0 – stop operare, defecțiune

t_{h13} – întîrzierea dintre momentul pornirii pompelor după o cădere de tensiune (comună tuturor stațiilor)

Constante diverse

x_{w1} – dacă diferența de presiune PT9 - PT10 ≥ x_{w1} , se opreşte pompa P1, trecîndu-se pe regimul artezian
x_{w2} – numărul maxim permis de porniri ale pompei P1/oră
x_{w3} – pasul utilizat pentru schimbarea în trepte a setpoint-ului regulatorului RG2
x_{w4} – pasul utilizat pentru schimbarea în trepte a setpoint-ului regulatorului RG12

2. Constante pentru stația de pompare

Constante pentru evaluarea stării pompelor

c_{p1} – numarul de ore de funcționare pentru pompa de forță P3
c_{p2} – numarul de ore de funcționare pentru pompa de forță P4
c_{p3} – numărul de porniri/oră ale pompei P3
c_{p4} – numărul de porniri/oră ale pompei P4

Constante de timp

t_{p1} – întîrzierea permisă după detectarea unei defecțiuni la transmițătorul PT1, variabila de process pentru regulatoarele RG11 şi RG12

t_{p2} – pasul de timp utilizat în schimbarea set-point-ului regulatorului RG3

t_{p3} – întîrzierea de la pornirea pompei P3 sau P4 după care se începe schimbarea setpoint-ului regulatorului RG3

t_{p4} – întîrzierea permisă după semnalarea alarmei PT1 ≥ HH şi oprirea pompei P3 sau P4

t_{p5} – întîrzierea permisă după semnalarea alarmei LT1 ≤ L şi oprirea pompei P3 sau P4

t_{p8} – întîrzierea permisă după semnalarea alarmei PT1 ≤ L şi pornirea pompei P3 sau P4

t_{p11} – întîrzierea permisă de la punerea ieşirii pe 0% a regulatorului RG3 şi oprirea pompei P3 sau P4

t_{h13} – întîrzierea dintre momentul pornirii pompelor după o cădere de tensiune (comună tuturor stațiilor)

Constante diverse

x_{p1} – numărul maxim permis de porniri ale pompei P3 respectiv P4/oră
x_{p3} – pasul utilizat pentru schimbarea în trepte a setpoint-ului regulatorului RG3

3. Constante pentru punctul termic

Constante pentru evaluarea stării pompelor

c_{h1} – numărul de porniri/oră pentru pompa de circulație P5
c_{h2} – numărul de porniri/oră pentru pompa de circulație P6
c_{h3} – numărul de porniri/oră pentru pompa de circulație P7
c_{h5} – numarul de ore de funcționare pentru pompa de circulație P5
c_{h6} – numarul de ore de funcționare pentru pompa de circulație P6
c_{h7} – numarul de ore de funcționare pentru pompa de circulație P7
c_{h8} – numarul de ore de funcționare pentru pompa de P8
c_{h9} – numarul de ore de funcționare pentru pompa de P9
c_{h10} – numărul de porniri/oră pentru pompa de P8
c_{h11} – numărul de porniri/oră pentru pompa de P9

Constante de timp

t_{h1} – întîrzierea minimă permisă între schimbarea pompelor P8-9

t_{h2} – timpul maxim de operare continuă a pompelor P8-9

t_{h3} – întîrzierea permisă de la raportarea unei defecțiuni la transmițătorii a căror mărime este utilizată ca variabilă de process în cadrul regulatoarelor

t_{h4} – întîrzierea permisă după semnalarea alarmei PT1≤ LL

t_{h5} – întîrzierea minimă permisă între pornirea pompelor P5-7

t_{h6} – întîrzierea permisă după semnalarea alarmei PT4≥ HH

t_{h7} – întîrzierea permisă după semnalarea alarmei PT5≤ LL

t_{h8} – întîrzierea permisă după semnalarea alarmei PT6≥ HH

t_{h9} – întîrzierea permisă după ce (PT6 – PT3)≥ x_{h2}

t_{h10} – întîrzierea permisă pentru oprirea pompelor de circulație P5-7

t_{h11} – întîrzierea permisă după semnalarea alarmei PT1≤ LL (CV6)

t_{h12} – întîrzierea permisă după ce (PT8 – PT7)≥ x_{h8}

t_{h13} – întîrzierea dintre momentul pornirii pompelor după o cădere de tensiune (comună tuturor stațiilor)

Constante diverse

x_{h1} – calibrarea lui WT1 – metri vint/puls
x_{h2} – maxima permisă a presiunii diferențiale pe schimbătoarele de căldură pentru circuitul de încălzire
x_{h3} – numărul maxim de porniri/oră ale pompelor P5-7
x_{h4} – valoarea minimă pentru PT5
x_{h5} – valoarea maximă pentru PT5
x_{h6} – diferența maximă între numărul de ore de funcționare ale pompelor P8 şi P9
x_{h7} – numărul maxim permis de porniri/oră ale pompelor P8-9
x_{h8} – maximă permisă a presiunii diferențiale pe schimbătoarele de căldură pt. circuitul de apă caldă menajeră
x_{h9} – pragul de sus al valorii TT16 in ecuația pentru calculul lui TT20
x_{h10} – pragul de jos al valorii TT16 in ecuația pentru calculul lui TT20
x_{h11} – valoarea maximă pentru TT20
x_{h12} – constantă în ecuația pentru calcului lui TT20
x_{h13} – panta în ecuația pentru calcului lui TT20
x_{h14} – valoarea minimă pentru TT20