

Best Linux Application 2009

Prezentare proiect dezvoltat

Nume : Cristian AXENIE;

Adresa: Str Nicolae Alexandrescu Dr, Nr 1, Bl C2, Ap 13, **0742033253**

Universitatea „Dunărea de Jos”, Galați, România;

Galati, str. Domneasca, nr. 111, <http://www.fsc.ugal.ro>

Facultatea de Știința Calculatoarelor

Specializarea : Automatică și Informatică Aplicată

Absolvent promoția 2005-2009

E-mail : cristian.axenie@gmail.com

Web : <http://robotics.viviti.com>

Nume Proiect : Control Tolerant la Defecte pentru Roboți Mobili

Raport descriere :

Control Tolerant la Defecte pentru Roboți Mobili

Tema proiectului redă de fapt tema lucrării de licență și a constat în dezvoltarea unei aplicații de control (conducere) în timp real a unui robot autonom diferențial (proiectat și creat de mine) cu capacitate de tolerare a defectelor ce apar în operarea acestuia. Astfel urmărind aspecte precum controlul distribuit, autonomia robotului, extensibilitatea structurii și flexibilitatea implementării am dezvoltat o aplicație puternică care susține la nivel inferior comunicarea facilă cu hardware-ul, implementarea taskurilor de control și diagnoză în timp real, la nivelul intermediar și un nivel superior al comunicației cu alte entități din mediul de operare. Înainte de a trece la descrierea arhitecturii aplicației propriu zise și a aspectelor de interes pentru dezvoltarea utilizând mecanismele și facilitățile aduse de Linux, se impune prezentarea obiectivului principal al temei de dezvoltare propuse. Comportamentul tolerant la defecte în cazul robotilor mobili se referă la posibilitatea de a detecta și identifica în mod automat defectele care apar în structură și de a continua operarea în prezența defectelor pentru care se emit măsuri corective.

Încă din faza de proiectare a aplicației s-a luat în considerare faptul că resursele hardware (numărul de senzori) sunt limitate și totuși prin intermediul aplicației dezvoltate trebuie asigurată autonomia robotului și capacitatea acestuia de a se adapta defectelor care pot apărea în structura sa în timpul operării. În implementarea curentă a aplicației robotul trebuie să urmărească o traекторie parametrizată în timp, cu o eroare minimă (trajectory tracking) și defectele care pot să apară în sistem se manifestă prin variații ale parametrilor constructive (defecte de tipul : pană a roții sau denivelare periodică a roții - dezechilibrare). După cum am menționat anterior datorită limitărilor hardware (roțile robotului nu se pot totuși dezumfla și nici nu se poate atașa un corp de roată în timpul operării) am ales injectarea software a defectelor menționate. La nivel structural aplicația conține două taskuri concurente ce rulează în timp real. Primul task este cel de control al robotului, care calculează o comandă corespunzătoare bazându-se pe informația de feedback de la sistemul de odometrie (poziționare, senzori : encodere) la fiecare 200ms. Algoritmul de control de tip Sliding Mode este bazat pe o structură în cascadă redată în figura 1, unde buclele interne rulează la fiecare 50ms.

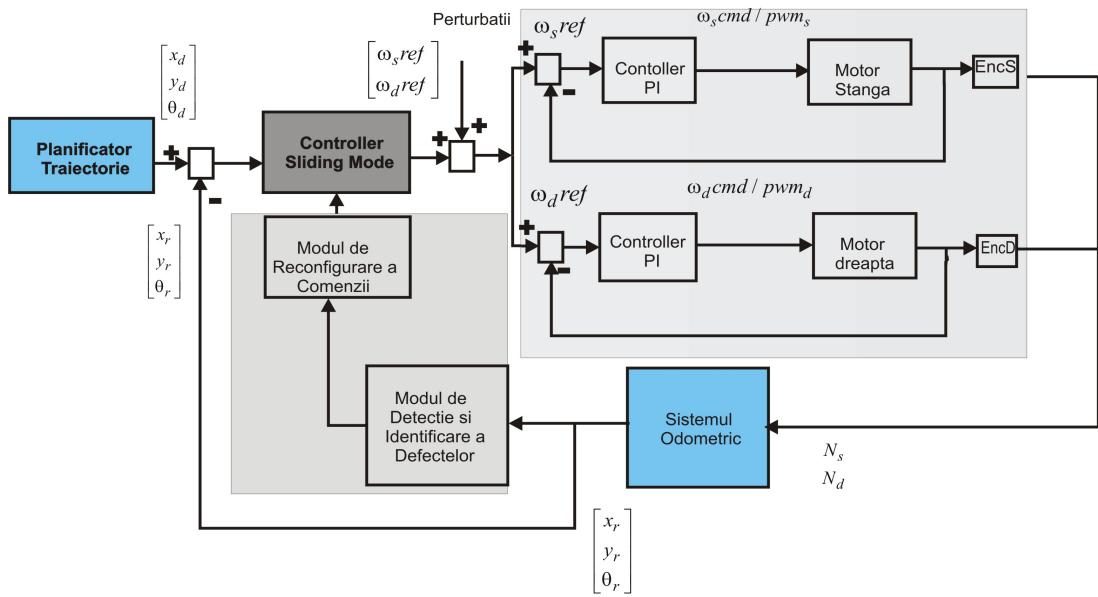


Figura 1 Bucla de control a robotului mobil

Al doilea task este cel de monitorizare și diagnoză care asigură detectia și identificarea defectelor (bazat pe măsurările de la encodere și predicții) și asigură suportul pentru reconfigurarea comenzi pentru robot în cazul apariției defectelor. În figura 2 este reprezentat sintetic modul de operare al modulului ce asigură capacitatea de toleranță a defectelor bazat pe 5 Filtre Kalman Extinse,

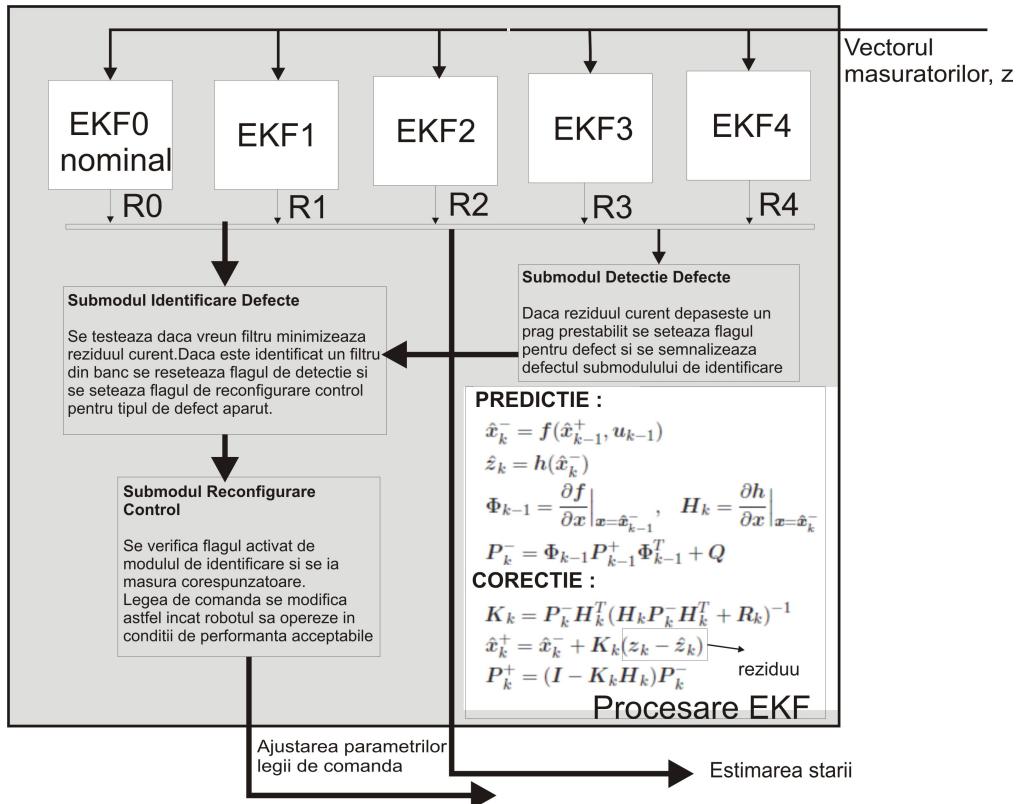


Figura 2 Modulul pentru toleranță la defecte

Pe scurt am realizat implementarea unui framework de injecție a defectelor, interactiv și distribuit, o structură cu cele 5 filtre recursive în care fiecare încorporează o copie a modelului cinematic al robotului dar cu parametri diferiți (fiecare filtru încorporează modelul robotului cu un anumit defecte și realizează predicții corespunzătoare operării cu defectul respectiv). Metoda se bazează pe calculul continuu al reziduurilor (diferența dintre valoarea măsurată și cea predictată) și verificarea depășirii pragurilor prestabilite (thresholding), deci detectie și identificare a defectelor bazată pe redundanță analitică care e soluția în cazul în care nu avem senzori redundanți disponibili. Concret, un defect este identificat în momentul în care reziduul depășește anumită valoare impusă (prag) și astfel filtrul care va genera reziduul minim va fi cel care determină identificarea defectului (reziduul minim al unui filtru marchează identificare defectului care este încorporat în modelul pe care se bazează filtrul).

În continuare sunt prezentate acele aspecte de interes strâns legate de utilizarea LinuxOS în aplicația embedded de control în timp real a robotului mobil. În ceea ce privește sistemul robotic mobil dezvoltat atributile de timp real se regăsesc în partea de control și monitorizare, în sensul că aplicația software dezvoltată are rolul de a asigura operarea robotului în contextul unor restricții temporale și de siguranță pre-stabilite. În implementare caracteristicile de timp real au fost introduse la nivelul sistemului de operare ce rulează pe platforma de prelucrare a datelor de pe robot. Astfel utilizând o platformă Intel bazată pe un procesor x86 Celeron am instalat o distribuție standard de Linux OS și anume RedHat9 pe kernel 2.4.24 (versiune veche pentru compatibilizare RTAI) peste care am aplicat patchul RTAI (Real Time Application Interface) pentru obținerea de facilități hard real time (abordare dual-kernel). În cazul de față am adăugat un nucleu de timp real distribuției Linux existente fără a-i altera funcționalitatea și bazându-se pe mecanisme de tipul virtualizării întreruperilor și unui mecanism specific pentru IPC între Linux și micro-nucleul de timp real. În abordarea considerată micro-nucleul de timp real se inserează între Linux și hardware, are un planificator separat și nu depinde de secțiunile critice al Linux. La apariția unei întreruperi micro-nucleul o capturează pentru a o utiliza în rutinele sale de timp real înainte de Linux, care va recepționa doar o întrerupere virtuală, făcându-l astfel un domeniu de prioritate secundară. Micro-nucleul are timpi de comutare între contexte foarte mici, cu o latență sub 20 µs și are acces la toată funcționalitatea domeniului Linux, nemijlocit. La baza mecanismului de bază a abordării RTAI se află ADEOS (Adaptive Domain Environment for Operating Systems) care permite partajarea resurselor hardware între mai multe sisteme de operare concurente. La bază, ADEOS este un nivel de abstractizare al resurselor disponibile ca un patch peste nucleul de bază Linux, care permite existența mai multor sisteme de operare (domenii) pe aceeași mașină. Pornind de la capacitatele sale de virtualizare ADEOS poate oferi o interfață de programare generică pentru domeniile supervizate care este independentă de arhitectura mașinii pe care operează. RTAI este alcătuit din mai multe componente, astfel prima componentă este nivelul de abstractizare al hardwareului (HAL) care oferă o interfață pentru accesul la hardware și care redă suportul funcțional pentru Linux cu capacitați hard real time. A doua componentă este nivelul de compatibilitate Linux care oferă o interfață către sistemul de operare Linux, și astfel oferind RTAI posibilitatea de a fi integrat în managementul taskurilor Linux, fără a influența operarea Linux. A treia componentă este nucleul de operare în timp real care introduce funcționalitatea hard real time pentru planificarea taskurilor, tratarea întreruperilor și securitate. A patra componentă este LX/RT care oferă suport pentru soft și hard real time în spațiul utilizator printr-o interfață de programare (API) pentru a oferi o funcționalitate similară apelurilor de funcții din spațiul kernel și din spațiul utilizator și un IPC simetric pentru cele două moduri. Ultima componentă a RTAI sunt pachetele de funcționalitate

extinsă , care cuprind drivere, interfețe de programare pentru diverse dispozitive, watchdogs software. Următoarea componentă software utilizată peste RTAI+Linux RTOS este driverul pentru placa de achiziție NI-6024E PCI pentru operații I/O real time COMEDI (Linux Control and Measurement Device Interface). Comedi este colecția de drivere pentru o mare varietate de plăci de achiziție de date, driverele având un nucleu comun pentru funcționalitate generică și module individuale de nivel scăzut specific fiecărui dispozitiv. Comedilib este o librărie de funcții destinață spațiului utilizator, pentru programarea aplicațiilor, configurare și calibrarea dispozitivelor. Kcomedilib este un modul kernel care oferă aceeași interfață oferită de Comedilib în spațiul utilizator, în spațiul kernel, fiind indicată pentru taskuri real time. În continuare este redată arhitectura aplicației dezvoltate,

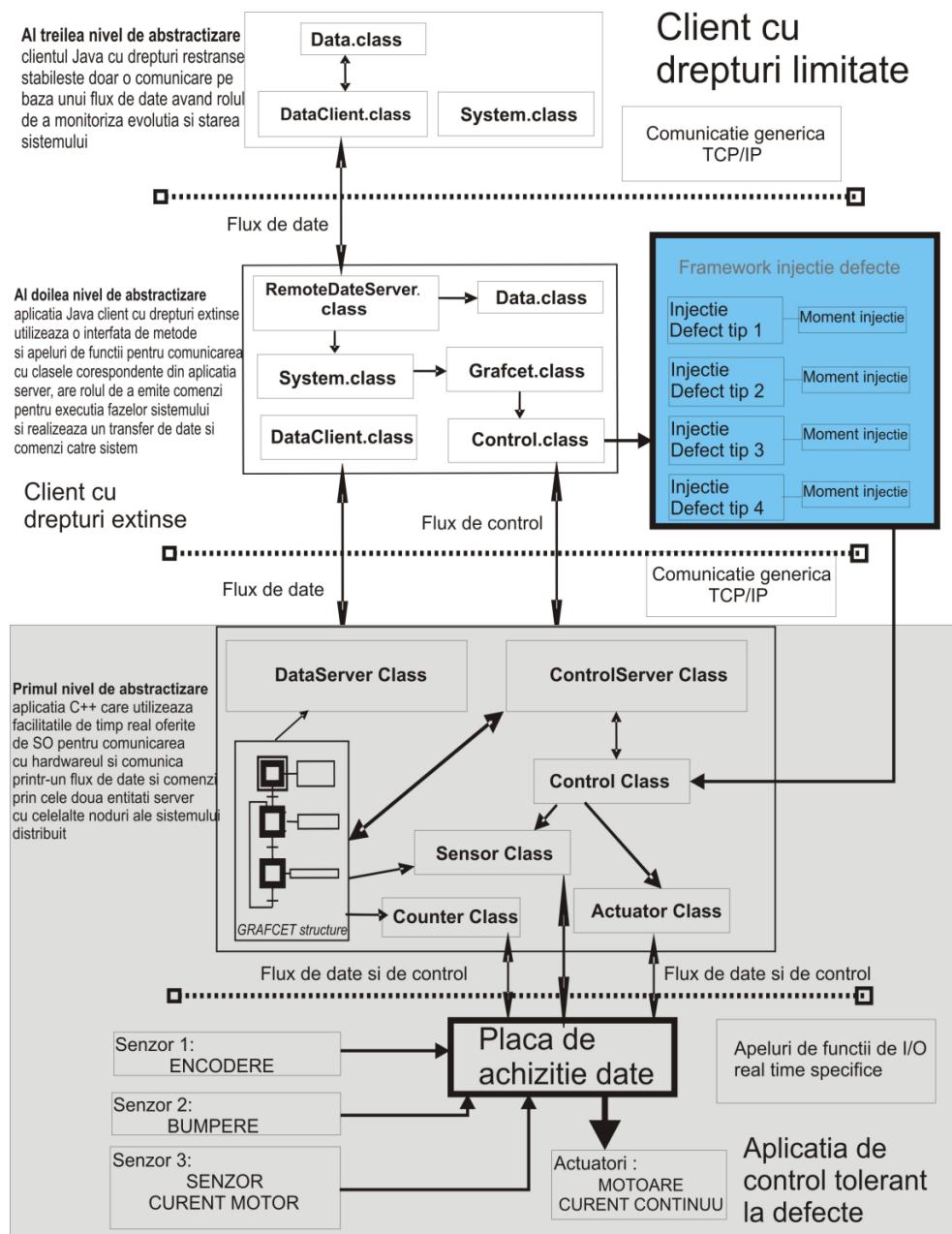


Figura 3 Arhitectura aplicației de control al robotului mobil

Aplicația este distribuită și la momentul curent doar clientul cu drepturi extinse (Java jar), care are rolul de a iniția operarea și de a implementa frameworkul de injecție al defectelor, este implementat. Aplicația server ce rulează pe mașina embedded de pe robot are o structură orientată obiect (C++) și permite abstractizarea în clase a elementelor specifice structurii fizice a robotului (senzori, actuatori) prin utilizarea funcțiilor specifice din API-ul Comedi. Astfel am dezvoltat pentru fiecare tip de senzor/ actuator, metode accesore de tip get() sau set() care apelează funcții Comedi. Clasa control are rolul de a implementa cele 2 taskuri de timp real (control + monitorizare) utilizând funcții specifice API-ului RTAI, pentru crearea, sincronizarea și managementul taskurilor real time. Prioritizarea și mecanismele de IPC au fost create manual în cadrul aplicație sunt forma utilizării unor metode ciclă care se apelează la fiecare perioadă de eşantionare și care introduc overhead datorită unor procesări intensive computațional. Pentru nivelul superior de comunicație cu celelele noduri ale sistemului distribuit am implementat două clase server de date și de control pentru transferul bidirectional de date (de la senzori, parametri initializare operare) și comenzi (start, stop, pause) clase care se bazează pe sockets și un link TCP/IP wireless.

În continuare este redată o descriere extinsă a detaliilor de implementare ale aplicației. Interfața cu hardware-ul, în spatele senzorii și actuatorii robotului, se realizează la nivelul aplicației server ce rulează pe robot. Un scenariu tipic de funcționare pornește cu citirea informațiilor de la senzori și în funcție de valorile citite se calculează o comandă pentru actuatori, citirile și comenzile fiind efectuate prin intermediul plăcii de achiziție.[60] Abstractizarea relațiilor cu senzorii și actuatorii s-a concretizat în designul aplicației prin crearea claselor Counter, Sensor și Actuator. Astfel la nivel funcțional clasa Sensor are rolul de a oferi un mijloc de a accesa valorile de la senzorii robotului. Datorită faptului că rețeaua de senzori este eterogenă clasa generică a fost extinsă cu două clase specifice pentru lucrul cu senzori care oferă informație analogică sau senzori care oferă informație digitală. Astfel la nivel funcțional clasa generică Sensor se bazează pe o funcție accesore de tip get() care este apoi redefinită în subclase. În cazul în care senzorul conectat oferă informație analogică la ieșire acesta este definit în aplicație ca un obiect AnalogSensor, care încapsulează informație despre dispozitivul Comedi la care este conectat, canalul, plaja de valori ale semnalului și o referință de tensiune. Întrucât senzorii care se pot conecta pot măsura diverse mărimi fizice funcția get() citește informația de pe canalul selectat cu un apel al funcției Comedi comedi_data_read() apoi apelează o funcție specială care realizează de fapt o conversie din mărimea fizică măsurată în tensiune, physicalDataToVolts(), care are rolul de a stabili dependența de pe caracteristica statică între mărimea de intrare și tensiune. În cazul de față am creat două instanțe ale clasei AnalogSensor, pentru interfațarea cu senzorul de curent de pe driverul de putere pentru motoarele de curent continuu, cu rol în monitorizare și siguranță și pentru monitorizarea tensiunii pe baterie. În cazul în care senzorul conectat oferă o informație digitală se instanțiază clasa DigitalSensor. Astfel obiectul creat încapsulează informații despre canalul plăcii de achiziție unde este conectat, identificatorul dispozitivului Comedi și informație specifică de configurare pentru intrare sau ieșire. Pentru a accesa valorile primite de la senzor se utilizează o metodă accesore get() care returnează valoarea binară primită de la senzor. În cazul robotului am creat două instanțe senzor digital pentru cele două bumpere, frontal și spate, care în starea necomprimată dau 1 logic pe intrarea plăcii, iar la comprimare comută în 0 logic, lucru care este captat de program și se interpretează ca o măsură de siguranță dând comandă PWM 0 către motoare. Deși face parte funcțional din clasa Sensor, clasa Counter a fost dezvoltată în paralel ca o clasă distinctă pentru interfațarea cu encodele. Astfel un obiect Counter încapsulează informații privind dispozitivul Comedi (un câmp generic utilizat la toate clasele pentru a asigura compatibilitatea cu ierarhia Comedi prezentată anterior) și canalul pe care este conectat

encoderul. Datorită faptului că placa de achiziție dispune de două intrări counter implementate hard, nu a mai fost nevoie de a dezvolta soft funcții speciale de numărare a trenurilor de impulsuri provenite de la encodere. Pentru a asigura un control eficient al operației de citire al encoderelor robotului am implementat funcții de reset, pentru a preveni overflow-ul counterelor de 12 biți, funcție de setare a direcției de numărare și a setării punctului de 0 pentru encoderul incremental și în final o funcție accesor getCount() pentru a prelua numărul de impulsuri din bufferul counterului în aplicație și pentru a utiliza această valoare la calculele de odometrie, pentru determinarea poziției robotului. Pornind de la ideea construirii unei structuri minime de robot mobil, sistemul senzorial este limitat dar suficient pentru a asigura controlul și a implementa facilitățile de redundanță analitică pentru diagnoză. În ceea ce privește sistemul de locomotie comenzi către actuatori sunt emise prin intermediul funcțiilor clasei Actuator. În acest caz similar cu senzorii, putem avea două tipuri de actuatori, actuatori comandați prin semnale digitale (DigitalActuator) sau actuatori comandați prin tensiuni continue (AnalogActuator). În cazul actuatorilor analogici obiectele corespunzătoare încapsulează informații despre canalul pe care sunt conectați, gama de valori pentru comanda admisibilă și o metodă accesor de tip set() cu rolul de a emite o comandă către actuator utilizând funcții din interfață de programare Comedi. Astfel după ce un obiect actuator a fost creat se dorește comanda acestuia, astfel se apelează funcția set(), care are rolul de testă că valoarea trimisă ca parametru este în gama de valori admisibile, se utilizează o funcție de tipul voltsToPhysicalData care are rolul de a asigura compatibilitatea între comanda în tensiune și intrarea actuatorului, apoi se utilizează o metodă de tip comedi_data_write() care scrie valoarea comenzi în bufferul de ieșire pentru un anumit canal al plăcii de achiziție și este emis către actuator. În cazul în care avem un actuator digital se instantiază clasa DigitalActuator specificând informații privind dispozitivul Comedi, canalul de conectare al actuatorului și informația de configurare pe intrare/ieșire. Pentru a emite comanda se apelează funcția set() specificând valoarea digitală care se trimit. În structura robotului există doi actuatori analogici, motoarele de curent continuu, care sunt comandate PWM.

În acest caz la nivelul aplicației se calculează comanda sub forma unei tensiuni analogice, care este emisă pe canalul corespunzător și preluată de driverul de putere care transformă această tensiune continuă în semnal PWM (Pulse Width Modulation) care este trimis motoarelor. În continuare este prezentată arhitectura internă a aplicației server ce rulează pe robot, urmând că în paragrafele următoare să se face o descriere a modului de funcționare al aplicației.

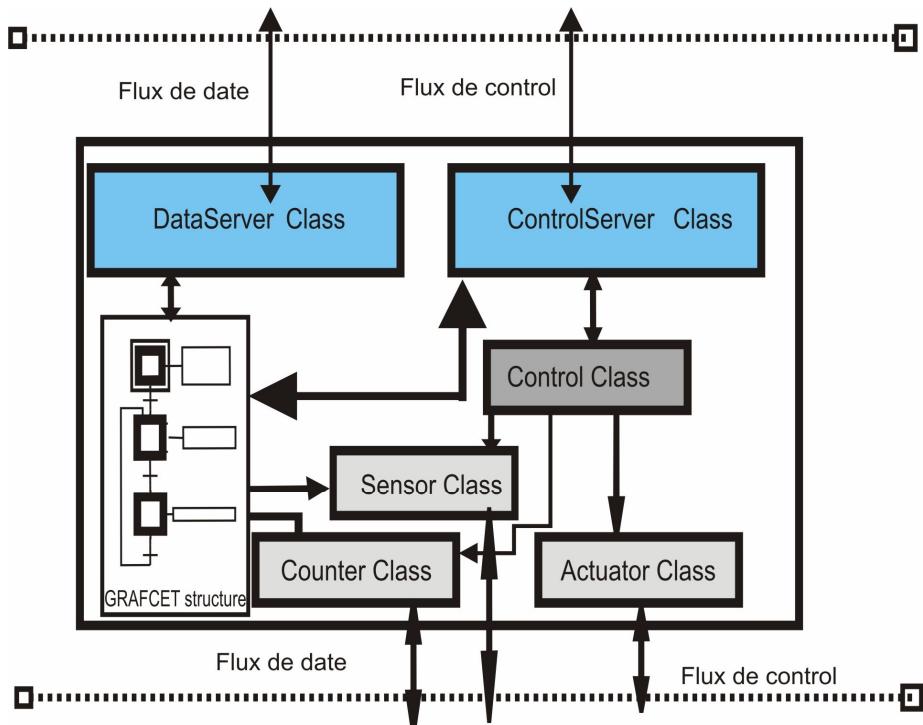
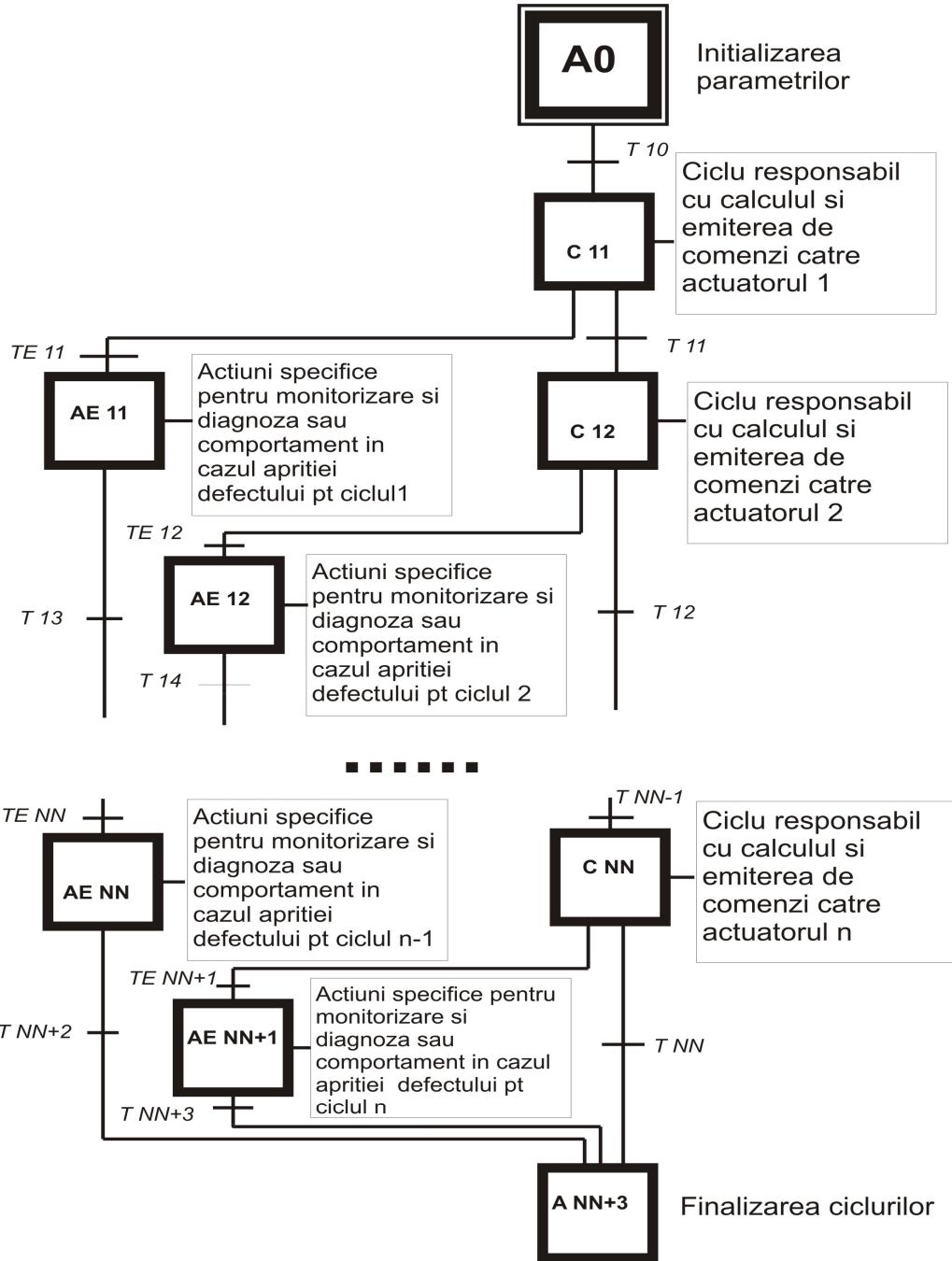


Figura4 Arhitectura internă a aplicației ce rulează pe robot

Pornind de la arhitectura aplicației primului nivel se trece în continuare la prezentarea suportului pentru implementarea algoritmului de control și diagnoză a defectelor.

Studiul nivelelor funcționale diferă de studiul nivelelor concrete de implementare în sensul că suportul pentru implementarea controlului și diagnozei se realizează concret prin conlucrarea celor două nivele ale aplicației, aplicația server C++ și aplicația client Java. La nivelul aplicației ce rulează pe robot se implementează efectiv taskurile de control și diagnoză, însă setarea parametrilor specifici de operare și activarea fazelor se realizează la nivelul aplicației client. Pentru a descrie aplicația la nivelul robotului se impune prezentarea aspectelor specifice ale instrumentului utilizat pentru execuția taskurilor, care redă de fapt modul de evoluție dorit al sistemului. Grafcet (Graphe Fonctionnel de Commande Etape - Transition) este un model folosit foarte mult pentru sistemele discrete logice și a devenit unul din instrumentele de bază pentru modelarea sistemelor de conducere bazate pe evenimente. Elementele specifice Grafcet au fost abstractizate și s-au concretizat sub forma unor clase. Clasa Stage modelează o etapă activă iar operarea sistemului se concretizează sub forma unei succesiuni de etape care redau de fapt trajectoria de stare a sistemului. Se precizează pentru fiecare etapă acțiunea care se va efectua, doar atunci când etapa este activă. Acțiunile pot fi însoțite de condiții care vor determina execuția acțiunii doar dacă condiția este îndeplinită și etapa este activă. Clasa Stage are ca subclase clasele Action, Cycle și Phase. Schimbarea comportamentului sistemului este marcată prin tranzitii de la o etapă la alta, o tranzitie fiind validată dacă toate etapele precedente sunt activate și dacă condiția logică (receptivitatea) este validată. Pentru a descrie mai bine funcționalitatea claselor specifice Grafcet sunt descrise cele două posibilități de execuție ale taskurilor asociate cu operarea sistemului.



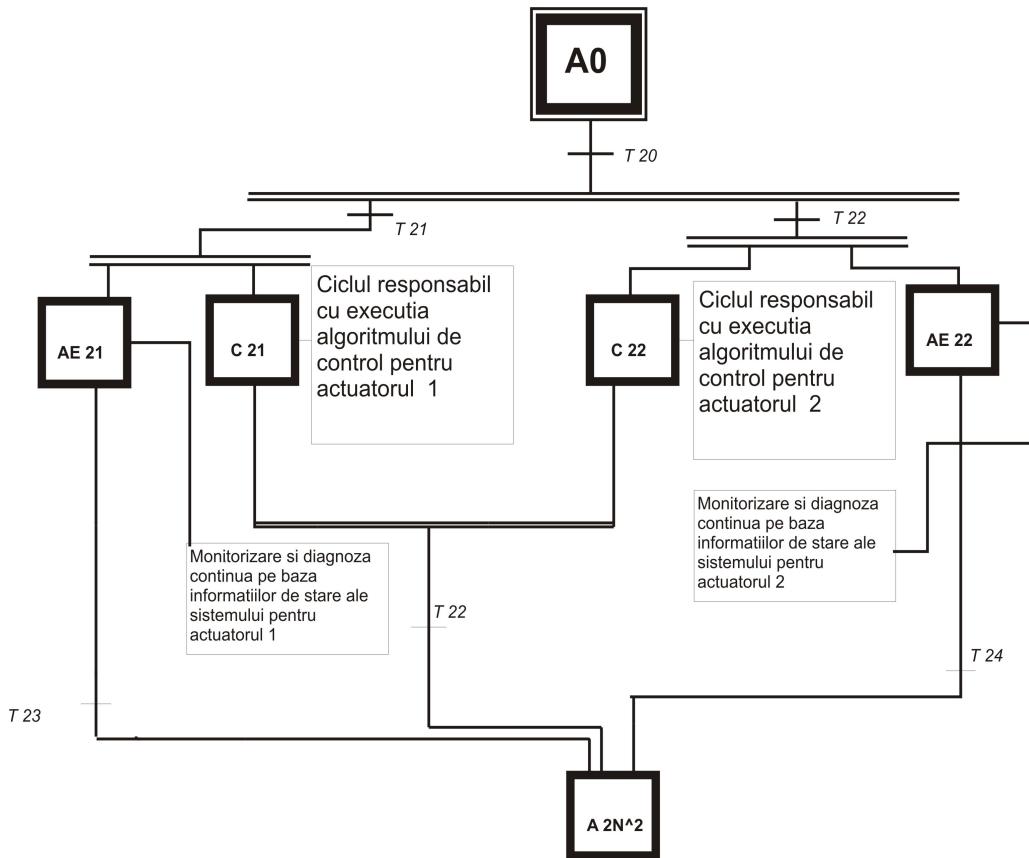


Figura 6 Structura generică pentru execuția etapelor în faza paralelă

Pornind de la structura generică prezentată în figura anterioară în implementarea curentă am introdus calculul comenzi cu algoritm sliding mode pentru ambele motoare (ambii actuatori) în același ciclu, iar în paralel cu execuția taskului de control are loc și o monitorizare continuă a comportamentului robotului pe baza informațiilor de stare de la senzori. Astfel prima etapă este inițializarea parametrilor robotului, parametrilor de control și parametrilor pentru modulul de detecție, apoi identificarea și în final reconfigurarea comenzi în prezența defectelor. După activarea tranziției inițiale se activează etapa de execuție paralelă a ciclului pentru control și a ciclului pentru monitorizare și diagnoză. Apoi prin activarea tranzițiilor finale se activează execuția etapei de final. O altă clasă importantă a aplicației server este clasa Control care are rolul de a seta parametrii pentru execuția etapelor Grafctului și are acces prin funcții specifice de get(), set() la parametrii de execuție și control, implementează interfață exportată către aplicația Java pentru controlul execuției prin metode de tipul exe_phase(), stop() și are un rol important și în stabilirea arhitecturii sistemului, cu privire la numărul și tipul senzorilor și actuatorilor (tabele de senzori și actuatori). Clasa Control este responsabilă și de managementul operațiilor de timp real utilizând apeluri de funcții specifice din interfața de programare a RTAI și Comedi. Prezentarea sintetică a facilităților și funcțiilor specifice RTAI utilizate poate fi realizată prin figura următoare.

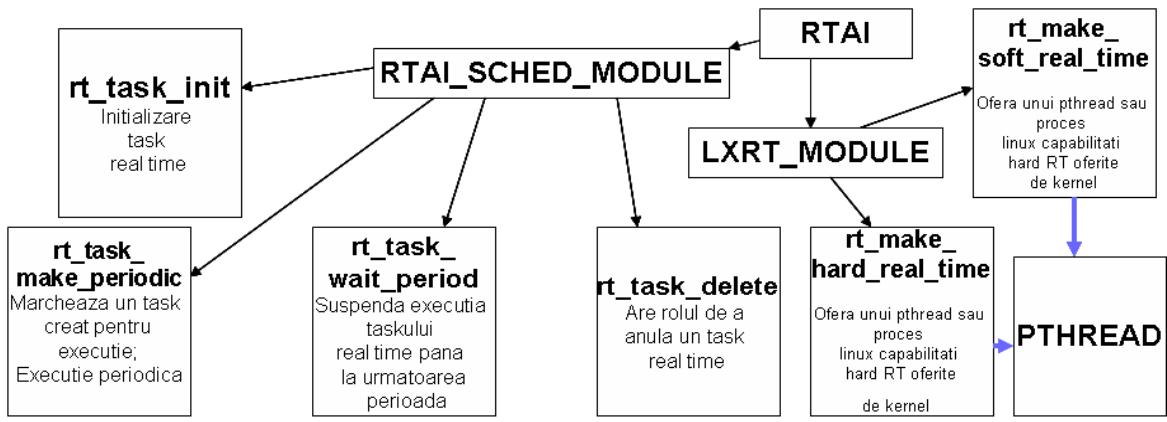


Figura 7 Facilități de lucru în timp real oferite de RTAI utilizate în aplicație

Înainte de a trece la următorul nivel al aplicației mai sunt prezentate câteva aspecte legate de implementare. Astfel clasa Thread oferă suportul pentru execuția în mediu multithreading, prin utilizarea unor metode specifice de tip debug () (cu rolul de a determina o localizare mai eficientă a erorilor în cazul executiei unei etape sau efectuarii unei tranzitii), metode de tip setName() sau getName() cu rolul de a seta/returna numele threadului activat și utilizează metode de asigurare a siguranței în mediul multithreaded. Clasa Action este o subclasă a clasei Thread și oferă facilitatea de emitere a comenzi de control pentru elementele de execuție (metode de identificare formală a unei acțiuni și comanda specifică), are loul de a interpreta etapele grafctului. Clasa Stage extinde clasa Thread și implementează conceptul de bază al conducerii prin metoda Grafct, implementând mecanisme pentru managementul (add(), start(), stop()) etapelor. În final se pune accentul pe implementarea clasei Cycle care determină de fapt lucrul cu facilitățile hard real time aduse de soluția aleasă. Astfel în alcătuirea ei se pot distinge metoda de tipul run() care folosește metode specifice inițializării agentului de timp real și a creării taskului corespunzător folosind metode specifice RT_SCHED+Pthread și metode de setare/returnare a perioadei de execuție cu evaluarea condițiilor de finalizare a execuției și managementul taskurilor de control al buclelor. Descrierea clasei Cycle se va realiza și ulterior în descrierea modului de implementare a celor două taskuri de control și diagnoză.

Nivelul de comunicație asigură schimbul fluxurilor de date și comenzi între nodurile sistemului distribuit. În ceea ce privește implementarea nivelului de comunicație între aplicații s-au implementat două clase server, ControlServer și DataServer. Clasa ControlServer are rolul de a transfera pachete de date care conțin comenzi de la client, în ceea ce privește activarea sau sistarea activității sistemului, iar clasa DataServer are rolul de a asigura un flux continuu de date pentru transferul parametrilor robotului către aplicația client. La nivel de implementare clasa ControlServer se bazează pe lucrul cu socketuri și la crearea unui obiect se utilizează informații privind adresa IP a clientului și portul de comunicație. Metodele specifice clasei sunt metodele de tip connect(), care așteaptă în background conectarea unui client și returnează o notificare, metode de tip serve_requests() care au rolul de a preluă informația din pachete și de a o transforma în comenzi explicate sau cereri către celelalte clase, metode de tipul serve_client() în care în funcție de comandă primită de la client se accesează anumite informații sau se activează execuția anumitor activități (start robot, stop robot) care de fapt redau legătura prin intermediul clasei Control cu elementele specifice de implementare ale aplicației de control. Serverul de date este implementat prin clasa DataServer și are rolul de a construi buffere pentru a stoca informații legate de evoluția sistemului în operare și de a le trimite aplicației client. Sunt utilizate metode care asigură conectarea mai mulțor clienți connect()

și metode specifice de transmitere de date, generic intitulate send(), care pot implementa transferul datelor formatare din buffere realizându-se și un management al cererilor de la clienti. În continuare este redată arhitectura clientului Java accentul căzând pe clasele care asigură comunicația cu robotul.

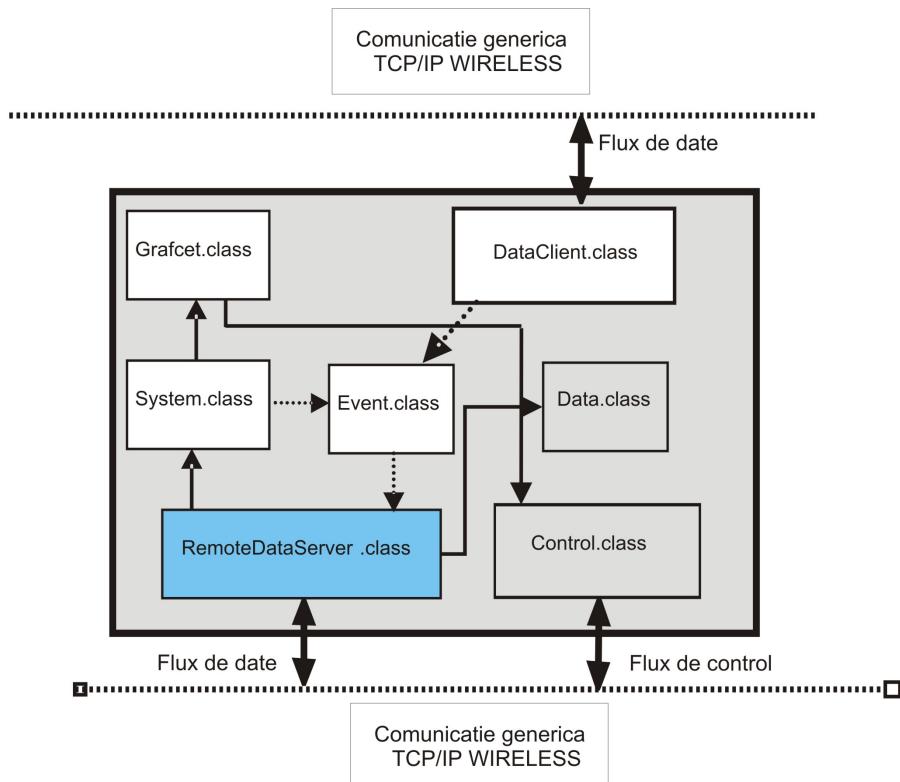


Figura 8 Arhitectura aplicației client Java

Aplicația client Java are o structură asemănătoare cu cea a aplicației de pe robot pentru a asigura compatibilitate la nivel funcțional. Astfel făcând referire la nivelul de comunicație, au fost implementate clasele RemoteDataServer, care comunică printr-un flux de date cu clasa DataServer a aplicației ce rulează pe robot și are rolul de a prelua informațiile de stare și parametrii curenti ale mărimilor de interes pentru robot și de a le transmite local clasei System și clasei Data. Mai departe clasa Data are rolul de a prelua și a stoca informațiile primite cu scopul de a emite loguri și alte surse de observație valide ce descriu întreaga perioadă de operare a robotului. Clasa System are rolul de a îngloba structura actuală a sistemului, numărul de senzori și actuatori și tipul lor pentru a putea prelua și transmite valorile parametrilor din bufferele de receptie de la robot către clasa Grafct. Clasa Grafct este similară cu cea dezvoltată în aplicația server cu diferența că aici are loc stabilirea efectivă a structurii Grafctului pentru execuție. După stabilirea parametrilor și structurii Grafct pentru execuție parametrii sunt trimiși clasei Control care comunică cu clasa ControlServer printr-un flux de comenzi care se reflectă de fapt în comenzi individuali de operare ale robotului. Clasa DataClient are rolul de a asigura comunicarea cu clientul cu drepturi restrânse, care nu a fost implementat în faza actuală. În implementarea actuală clientul Java este construit pe baza unei interfețe utilizator GUI sub forma arhive executabile Java (jar) care are rolul de a activa sau opri activitatea robotului și de a injecta defectele în cadrul operării tolerante la defecte. Datorită faptului că nu a fost

posibilă introducerea unor defecte on-line, în timpul operării robotului a fost aleasă dezvoltarea unei metode de injectie software a defectelor prin alterarea anumitor parametrii ai modelului matematic al robotului. Astfel pe lângă comenziile transmise către serverul de control al aplicației server se transmit pachete care conțin variabile indicatori pentru declanșarea anumitor defecte în sistem. O altă posibilitate oferită de aplicația client este de a realiza un control la nivel de execuție pas cu pas a algoritmului de control, însă în cazul de față nu se impune. Alegerea tipului de execuție a taskurilor (serială / paralelă) se poate realiza direct din interfața utilizator și de asemenea se pot încărca fișiere care să introducă o structură Grafcat diferită în funcție de problemă. În acest caz însă taskurile de control și monitorizare și diagnoză sunt concurente și există posibilitatea de a monitoriza grafic evoluția anumitor variabile ale sistemului, accesând informațiile stocate în clasa Data, provenite de la robot(opțiune încă în lucru). Injectia defectelor nu se poate realiza în timp real datorită priorității mari a taskului de control pe mașina de pe robot și datorită faptului ca nu e preemptibil am ales o abordare mai deterministă , în sensul că alegerea momentelor (arbitrare) de injectie a defectelor și tipul defectului se realizează înainte de pornirea robotului și astfel nu se pierde nici din validitatea metodei.

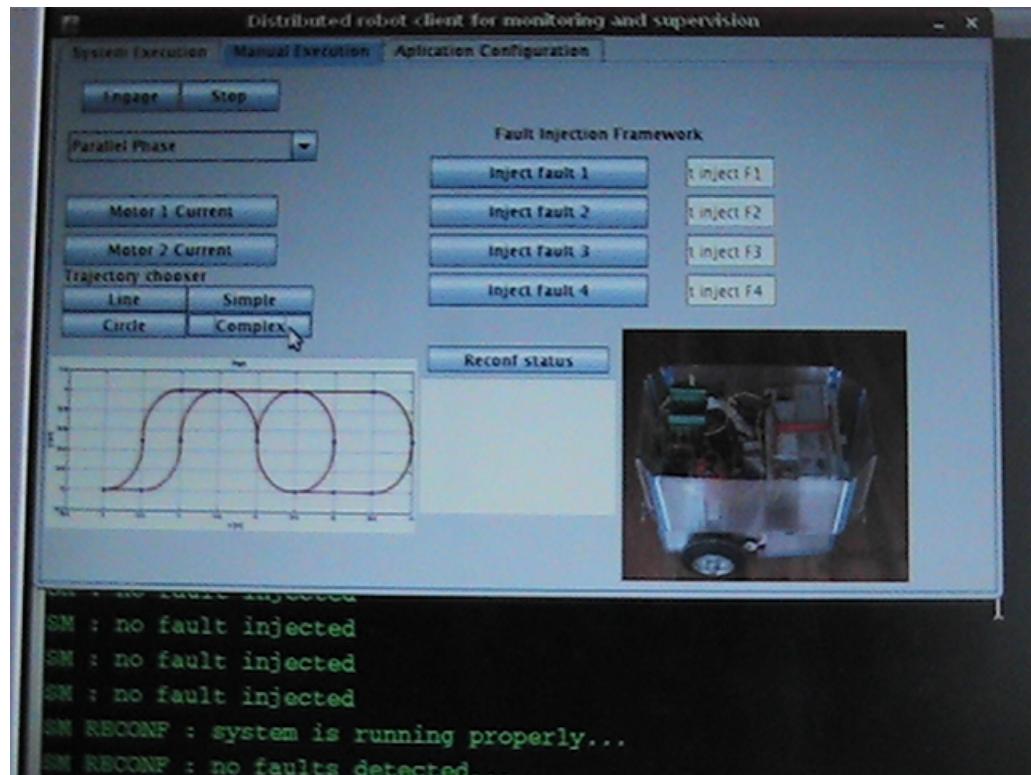
Măsurarea performanțelor obținute a fost marcată prin analiza răspunsului robotului pentru diverse traiectorii și care marchează în fapt minimizarea latenței globale a sistemului la calculul unei noi comenzi la fiecare perioadă de eşantionare. Întradevăr comparativ cu o soluție simplă bazată pe MCU, overheadul introdus de prezența Linux + RTAI RTOS e semnificativ și e motivat doar de eficiență și puterea de calcul a unei mașini pe 32 de biți în calculul concurrent al comenzi către robot și a calculului matriceal recursiv al filtrelor de monitorizare și diagnoză.

Ca future work (da fapt current work) lucrez la portarea aplicatiei pe o platformă PowerPC MPC8315ERDB utilizând un RTOS mai avansat numit Xenomai (superset al RTAI, cu performante real time superioare) și în acest moment colaborez cu un dezvoltator francez pentru portarea Comedi către noul RTOS și astfel portarea întregii aplicări dezvoltate pe noua platformă.

Proiectul de față nu vine ca un proiect care să îmbunătățească prin noi facilități kernelul Linux sau o anumită distribuție ci doar redă faptul că utilizând instrumente dezvoltate în comunitatea Open-Source (RTAI, Comedi, Linux RH9 (free)) am reușit să implementez o aplicație flexibilă și prin care încerc să pun în lumină importanța și eficiența care o introduce utilizarea Linux și în domeniul industrial și robotică.

- La momentul curent (28.08.2009) este disponibil codul întregii aplicații (client + server) pentru o platformă server bazată pe x86 (Intel Celeron) și în cazul în care până la atingerea deadline-ului voi finaliza dezvoltarea voi trimite și varianta de aplicație server pentru PowerPC.
- Dacă sunt de interes mai multe detalii despre realizare și rezultatele obținute (grafice, loturi de date) acestea pot fi făcute disponibile , fie pe web-site, fie prin e-mail.
- Pentru o demonstrație se pot urmări micile filmări de pe : <http://robotics.viviti.com>

Interfața aplicației client Java



Robotul

