Universitatea Politehnica Bucuresti

Facultatea de Electronica, Telecomunicatii si Tehnologia Informatiei

Proiectarea produselor software destinate a funcționa în timp real

Studenți: Ana - Maria Radu

Mădălin Nica

Dan Arama

Grupa: 442 A

Cuprins

1. Introducere

(Dan Arama)

În ultimii ani, programarea în timp real a devenit o parte importantă în inginerie şi informatică. Datorită puterii de calcul în creştere şi a implementărilor, software-ele (SW) sistemelor sunt într-o continuă optimizare în ceea ce priveşte flexibilitatea şi sofisticarea. Cu toate acestea, produsele SW în timp real devin din ce în ce mai complexe, metodele de proiectare jucând un rol important în dezvoltatea acestora.

În mod tradițional, comunitatea care dezvoltă produse SW este reticentă în adoptarea noilor tehnologii de proiectare din cauza performanțelor mari şi a timpilor stricți de răspuns necesari pentru majoritatea sistemelor, ceea ce îi obligă pe dezvoltatori să folosească tehnicile de bază. Alți factori care contribuie la păstrarea mentalității sunt costurile foarte mari şi cerințele de siguranță a multor astfel de sisteme. Aşadar, este preferată abordarea implementării produselor SW cu tehnologii de bază în defavoarea tehnologiilor noi, probabil mai bune, dar netestate.

Totuşi, în urma progreselor recente în tehnologiile moderne programate, se încearcă o combinare a celor două direcții în ceea ce priveşte proiectarea produselor SW. Calculatoarele devin din ce în ce mai rapide, iar tool-urile pentru implementarea software a sistemelor sunt din ce în ce mai puternice. În plus, tehnicile tradiționale de implementare încep să nu mai fie de ajuns pe măsură ce cerințele pentru funcționalitățiile mai sofisticate sunt într-o continuă creştere, iar produsele SW devin mai complexe. Prin urmare, pentru multe aplicații destinate a funcționa în timp real este necesară utilizarea soluțiilor de ultimă generație.

1.1 Definiții

Un **sistem in timp real** este un sistem software in care functionarea corecta a sistemului depinde de rezultatele produse de sistem si de timpul pe cu care acestea sunt produse.

Un **sistem soft in timp real** este un sistem in care operatiile sunt eronate daca rezultatele nu sunt indeplinite in timpul specificat.

Un **sistem hard in timp real** este un sistem in care operatiile sunt incorecte daca rezultatele nu sunt produse in timpul specificat.

Proiectarea sistemelor software (SW) destinate a funcționa în timp real (R-T) reprezintă implementarea sistemelor SW încorporate într-un alt sistem hardware mai mare, al cărui comportament este supus unor constrângeri de timp.

Un **sistem care funcționeaza în R-T** este un sistem al cărei functionalități depinde de rezultatele produse de sistem şi de timpul la care aceste rezultate sunt produse. Sistemele se pot clasifica dupa cerințele legate de timp în sisteme SW în R-T, pentru care erorile ocazionale legate de timpul de răspuns / execuție pot fi

tolerate, şi sisteme HW în R-T, ale căror eşuări de a îndeplini termenii limită de execuție a task-urilor sunt tratate ca şi o eroare catastrofică a sistemelor.

1.2 Caracteristici ale sistemelor SW în R-T

Termenul “timp real” se referă la abilitatea produsului de a executa anumite task-uri sau de a produce anumite rezultate exact în momentul când trebuie executate. Altfel spus, produsele SW care funcționează în R-T se refera la programele care pot executa sarcinile şi funcțiile asociate exact în momentul în care acestea au fost atribuite sistemului, şi nu dupa ce programul a fost executat sau după o anumită perioadă de timp programată. Programele SW în R-T rulează în mod automat dupa ce setările lor au fost configurate; de asemenea, ele se pot acomoda imediat cu anumite modificări făcute de programator sau user la un moment dat. Aşadar, timpul este caracteristica esențială a sistemelor care funcționează în R-T, iar activitățiile trebuiesc programate şi executate astfel încât să se respecte cerințele temporale.

O altă caracteristică fundamentală a sistemelor în R-T este faptul că sunt sisteme embedded (sunt încorporate într-un sistem mai mare care interacționează cu mediul înconjurător). Acest lucru reprezintă adeseori motivul fundamental al complexității sistemelor R-T. În general, mediul fizic are caracter nedeterminist, cu evenimente care se pot întampla asincron, concurente şi neprevazute, astfel că, oricare ar fi evenimentele petrecute, un sistem R-T embedded trebuie să răspundă corespunzător din punct de vedere al timpului şi al celor întâmplate.

Concurența mediului fizic implică de obicei ca şi produsele SW care funcționează în R-T să fie concurente. De fapt, una din principalele cerințe pentru multe sisteme R-T este sincronizarea evenimentelor concurente multiple pe măsură ce acționează asupra mediului înconjurător. Din nefericire, concurența măreşte complexitatea proiectării produselor SW din moment ce ea este în conflict cu serializarea în mod inerent, o problemă cauză-efect legată de interferența cu omul.

O altă caracteristică a acestor sisteme este dependența. Multe sisteme R-T joaca un rol crucial în mediile în care acționează, iar eşuarea de a produce rezultatele corecte poate rezulta la mărirea costurilor sau chiar de a pune în pericol siguranța umană. Ca un rezultat, sistemele în R-T trebuie să fie foarte de încredere (trebuie să efectueze corect anumite task-uri) şi disponibile (trebuie să efectueze task-uri în mod continuu).

2. Managementul concurenței

(Ana – Maria Radu)

O mare problemă pe care o au proiectanții produselor destinate sistemelor care funcționează în timp real o constituie managementul concurenței. Chiar şi în cazul sistemelor cu un singur procesor, concurența este adesea folosită atât pentru a simplifica structura produsui SW, cât şi pentru a îmbunătății răspunsul sistemului la evenimentele critice. Această formă de concurență se referă la multi-tasking, şi implică

multiplexarea sarcinilor concurente la un singur procesor. Totuşi, în timp ce concurența este un instrument necesar pentru a administra răspunsul în timp al unui sistem, ea implică şi dificultăți în proiectare. Î n general, manevrarea ineficientă a concurenței poate conduce la apariția unor bug-uri în SW care pot fi greu de găsit şi corectat, scăzând astfel performanțele întregului sistem. De asemenea, concurența face ca estimarea performanțelor sistemului să fie mai complexa.

Î n consecință, au fost dezvoltate tehnici speciale de management al concurenței pentru a asigura controlul acesteia. Acestea sunt bazate pe trei tipuri de mecanisme :

- Mecanisme pentru reprezentarea şi urmărirea progresului activităților concurente ;
- Mecanisme pentru comunicarea dintre sarcinile concurente ;
- Mecanisme pentru sincronizarea execuției activităților concurente multiple.

2.1 Activităţi concurente

Cel mai important mecanism pentru reprezentarea activităților concurente îl reprezintă alocarea fiecărei sarcini un procesor 'virtual' care să execute un fir de execuție de control şi care să îşi mențină starea în timpul execuției. Ne referim la acest procesor virtual ca fiind un proces. Î n mod normal, orice proces are propriul spațiu de adrese care este logic diferit de spațiul de adrese al altui proces. Din păcate, suprasarcinile necesită ca procesorul fizic să trecă de la un procesor fizic la altul, fapt care implică întârzieri generate de trecerea de la un registru la altul, chiar şi pentru procesoarele foarte rapide.

Pentru a reduce suprasarcina, multe sisteme de operare au posibilitatea de a include mai multe fire de execuție pentru un singur proces, care împart spațiul de adrese. Acest lucru reduce supraaglomerările, dar există posibilitatea de accesare simultană a aceluiaşi spațiu din memorie, ceea ce conduce la un conflict.

O altă variantă pentru managementul activităților concurente este existența unei aplicații de sine stătătoare care să manevreze concurența. Această tehnică este foarte comună în cazul sistemelor foarte mari, cu un numar ridicat de fire de execuție concurente.

2.2 Mecanisme de comunicare

Comunicarea datelor se bazează în general fie pe împărțirea datelor, fie pe transmiterea mesajelor. Î ntr-o arhitectură cu împărțirea datelor, două sau mai multe fire de execuție pot accesa datele, comunicând astfel între unul cu altul. Pe de alta parte, transmiterea de mesaje implică schimbul explicit de date dintre doua fire de execuție.

Atunci când se foloseşte partajarea datelor, accesul concurent poate conduce la modificări neconcordante. Din acest motiv, trebuie să existe un mecanism de sincronizare, numit excludere reciproca. Î n transmiterea de mesaje, comunicarea poate fi asincronă, cât şi sincronă. Î n cazul comunicației asincrone, activitatea de trimitere nu este sincronizată cu cea de recepționare ; în schimb, se formează anumite buffer-e, din care informația poate fi extrasă mai tarziu. Receptorul şi transmițătorul lucrează independent unul de altul, şi nu

se pot face presupuneri privitor la starea celuilalt. Comunicarea sincronă presupune sincronizarea dintre transmițător şi receptor pentru a se face transferul de informații.

2.3 Mecanisme de sincronizare

Sincronizarea implică coordonarea execuției dintre două sau mai multe activități în aşa fel încât conflictele privind concurența să fie evitate. Excluderea reciprocă este una din formele sincronizării. O altă variantă mai sigură şi eficientă este encapsularea datelor partajate ca date ale unui obiect protejat şi făcând datele accesibile doar prin operații ale obiectului respectiv. Î n acest fel, un singur fir de execuție are dreptul de a accesa obiectul protejat.

3. Paradigme de proiectare

(Madalin Nica)

Termenul 'paradigme de proiectare' se referă la metodele de proiectare ale produselor SW destinate aplicațiilor care funcționează în timp real şi în principal la modalitatea în care activitățiile concurente sunt reprezentate şi modelate. Î n continuare vom prezenta cele mai des folosite metode de proiectare.

- **Metodele de proiectare Jackson** [1](cunoscută sub numele de 'Programarea structurala Jackson') **şi Warnier** [2] sunt două metode de proiectare orientate pe structuri de date (data-structured-oriented), aplicabile în principal proiectării designului produselor SW. Nici una dintre cele două metode nu tratează problema decompoziției sistemului în module sau taskuri, şi în consecință, nici una nu este potrivită pentru proiectarea produselor SW destinate funcționării în R-T. În acest caz, structura sistemului computerizat este derivata din structura activitatii, exprimata printr-o serie de modele. Structura bazei de date este derivata din modelul informational, cel mai adesea exprimat prin entitati, atibute si relatii. Programul logic este derivat dintr-o serie de modelari ale proceselor. Crearea de prototipuri este uneori o alternativa a acestor metodolgii structurate, trecând peste necesitatea crearii de modele ale activitatii. Tehnica prototipurilor necesita întelegerea cerintelor si a structurii, care se expliciteaza cel mai bine prin creare de modele. De aceea, nu este o alternativa, ci un aspect complementar.

- **Designul structurat (Structured Design)** [3] [4] este o metodă de implementare a sistemelor care tratează împărțirea sistemului în module. Aceasta tehnică poate conduce la arhitecturi modulare cu mare funcționalitate, dar nu furnizează nici un ajutor în structurarea sistemului în taskuri. Î n plus, nu ia în calcul nici arhitectura modulelor interne ale sistemului. Designul Structurat este adesea folosit împreuna cu **Analiza Structurată (Structured Analysis)** [5] [6], care foloseşte diagramele fluxului de date şi împărțirea funcțională.

- **Metoda de proiectare a produselor SW cu ordin superior (Higher Order Software Design Method)** [7] foloseşte de asemenea decompoziția funcțională pentru împărțirea sistemului în module, dar eşuează, de asemenea, să rezolve problema împărțirii sistemului în sarcini concurente.

- **Metoda Mascot** [8] este o soluție potrivită pentru sistemele care funcționează în timp real. Ea tratează în mod specific structurarea sistemului în task-uri şi defineşte interfețele dintre acestea. Cu toate acestea, aceasta paradigmă începe cu diagrama conexiunii dintre sarcini şi nu ia in calcul problema a cum sistemul îşi împarte task-urile, nici structura individuală a sarcinilor în sine. Abordarea proiectarii sistemelor in timp-real folosind metoda MASCOT implica definirea unui cadru si a unor notatii grafice pentru a permite integrarea in timp-real a activitatii sistemului. O activitate este un proces care este independent de alte activitati si prin urmare poate fi planificat sa fie executat simultan cu alte activitati. Tehnica MASCOT ofera un set util de mecanisme de sincronizare si comunicare care permit dezvoltarea separata a modulelor individuale.

  Procedeul are doua reprezentari echivalente: **prin diagrame si textuala.** In continuare se va dezvolta in special forma de reprezentare prin diagrame si mai putin cea textuala. in locul celei din urma se prefera descrierea folosind retele Petri. Formalismul este folosit pentru determinarea structurii proiectului unui sistem soft. El serveste cu succes la testarea programelor in timpui fazei de dezvoltare si celei ulterioare ei.

  Metoda se concentreaza pe reprezentarea concurentei si a fluxului de date (engl.: *dataflow),* deoarece aceste caracteristici sunt prezente in mediul in care astfel de sisteme sunt inciuse. Un sistem de conducere este compus dintr-un numar mare de senzori, traductoare si eliemente de execute, generind, respectiv consumand date. Ele opereaza in aceiasi timp, prin urmare concurent.

  Reprezentarea prin diagrame sau reprezentarea textuala permit descrierea proiectului intr-o forma ierarhizata. Proiectul poate fi dezvoltat de sus in jos (engl.: *top-down),* de jos in sus (engl.: *bottom-up),* sau combinamd cele doua metode.

  Elaborarea programelor prin metoda MASCOT se divide in 6 etape.

  Etapa 1. Cerinte si constrangeri

  Se stabilesc cerintele generate si constrangerile externe,

  Etapa 2. Propunerea unui proiect

  La nivelul superior se construieste o propunere pentru un proiect, care sa respecte cerinteie si constrangeriie externe. Fiecare componenta este specificata prin termenii functiei sale si a modului in care contribute la proiectul general.

  Etapa Descompunerea sistemului

  Se continua cu elaborarea progresiva a eiementelor componente specificandu-se activitatile, ZDC-urile si serverii. Fiecare componenta trebuie precizata tinand cont de datele de irrteractiune. Se descriu porturile si ferestrele etementelor.

Etapa 4. Descompunerea elementelor

Proiectarea continua cu descompunerea fiecarei activity compuse identificate in etapa anterioara. Se precizeaza fiecare radacina si subradacina, interactiunile cu alte subradacini si porturile activitatilor.

Etapa *5.* Programarea

Se detaliaza specificatiile fiecarui tip de interfata, iar algoritmul fiecarui sablon este codificat foiosind interfetele definite.

Etapa 6. Integrarea si testarea

Se compileaza si se testeaza mai intai componentele simple, apoi se testeaza retelele. sabloaneie compuse se testeaza fiecare imparte folosind reteaua sa specifica.

- **Metoda Parnas a ascunderii informației (Parnas' Information Hiding Method)** [9] este un concept de proiectare puternic care conduce la sisteme foarte modulate cu puține elemente intermediare. Această tehnică a fost adoptată şi de celelalte metode de proiectare precum Designul Structurat, Mascot şi Modelul Obiect-Orientat [10].

- **DARTS (Design Approach for Real – Time Systems)** este o metodă extinsa din metodele Analiză Structurată şi Design Structurat. Această tehnică furnizează o abordare eficientă a structurării unui sistem care funcționează în timp real în task-uri, precum şi un mecanism de definire a interfețelor dintre aceste sarcini. Î n acest sens, se bazează pe experiențele şi rezultatele obținute cu celelalte paradigme.

4. Cerințele proiectării unui sistem care funcționează în timp real

(Ana – Maria Radu)

Fiecare dintre metodele de proiectare a produselor SW menționate anterior prezintă anumite limitări în ceea ce priveşte proiectarea sistemelor care funcționează în timp real. Cele mai potrivite tehnici care satisfac cerințele proiectării sistemelor în R-T sunt Design Structurat şi Mascot.

Î n continuare sunt prezentate principalele cerințe ale unei metode eficiente de proiectare a produselor SW destinate aplicațiilor care funcționează în timp real.

4.1 Proiectarea bazate pe fluxul de date

Abordarea fluxului de date la proiectarea produselor SW este foarte potrivită în contextul implementării sistemelelor care funcționează în R-T, deoarece datele din aceste sisteme pot fi vazute ca fiind transferate de la intrare la ieşire, iar la nivel intermediar, să fie transformate în sarcini software.

Metodele orientate pe fluxul de date sunt cel mai bine exemplificate în stilurile Analiză Structurată si Design Structurat, care în mod frecvent sunt utilizate împreună. Î mpreună cu Analiza Structurată, diagramele

fluxului de date sunt folosite pentru a arăta funcțiile (transformările) sistemului, precum şi fluxul de date dintre aceste transformări şi stocările de date accesate de acestea. Alte caracteristici ar fi utilizarea unui dicționar de date şi o decompoziție ierarhică a transformărilor.

Designul Structurat, cunoscut şi sub numele de Designul Compus (Composite Design) constă în două componente principale : (1) setul de criterii, coeziune şi cuplare, care sunt folosite pentru evaluarea calității proiectării, şi (2) metoda de proiectare pentru ghidarea arhitecțiilor în structurarea sistemului în module de sus în jos. Scopul acestei metode este de a produce un model în care modulele să fie se completeze unele cu altele şi să nu existe module intermediare.

4.2 Comunicarea dintre task-uri şi sincronizarea acestora

Deoarece este esențial ca în cazul sistemelor care funcționează în timp real sarcinile să comunice şi să-şi sincronizeze operațiile, multe sisteme de operare care funcționează în timp real prezintă anumite mecanisme pentru rezolvarea cele două probleme. Cele mai comune astfel de mecanisme sunt prezentate în cele ce urmează.

4.3 Sincronizarea task-urilor

Cele doua modalități de sincronizare a sarcinilor sunt excluderea reciprocă şi simularea încrucişată.

- Excluderea reciprocă este necesară atunci când datele partajate pot fi accesate de două sau mai multe activități concurente [11] . Această modalitate este posibilă prin implementarea unor ‘semafoare binare’.

- Simularea încrucişată are loc atunci când una din task-uri aşteaptă un semnal de la cealaltă sarcină înainte de a-şi începe execuția. Atât ‘semafoarele binare’ cât şi sincronizarea evenimentelor pot fi folosite pentru implementarea simulării încrucişate.

4.4 Comunicarea dintre task-uri

Î n general, comunicarea dintre task-uri se face prin transmiterea mesajelor [12], mecanism furnizat în trei modalități : de către sistemul de operare, prin furnizarea unui proces multitasking cu capacitate de comunicare a task-urilor implementate în limbajul de programare, ori cu ajutorul unui modul separat care se ocupă de transmiterea mesajelor, folosind primitivele de sincronizare oferite de sistemul de operare. Această abordare este utilizată în metoda Mascot, unde sarcinile comunică între ele prin intermediul altor canale. Canalele sunt folosite pentru transferul de date, aşa cum mesajele sunt pentru transferul taskurilor.

4.5 Ascunderea informației

Conceptul de ascundere a informației (cunoscut şi sub numele de abstractizarea datelor) a fost introdus de Parnas [9] ca şi un criteriu de descompunere modulară a sistemelor. Obiectivul este de a ascunde deciziile cheie de proiectare, astfel încât fiecare decizie cheie de proiectare trebuie cunoscute doar de către un singur modul. Prin ascunderea informației, cantitatea de informație împărtită între module este minimă.

Avantajul acestei metode este că fiecare modul conține informații diferite, fiind astfel mai bine controlate. Dezavantajul o constituie suprasolicitarea generată prin accesarea structurilor de date cu ajutorul unei funcții, mai degrabă decât direct.

4.6 Dependența de stare în procesarea tranzacțiilor

Majoritatea sistemelor care funcționează în timp real sunt orientate pe tranzacții sau încorporează o anumită cantitate de procesare a tranzacțiilor (de exemplu, acțiunea sau secvența de acțiuni care trebuie executată depinde de natura datelor de intrare).

5. Proiectarea produselor SW care funcționează în timp real

(Madalin Nica)

Proiectarea produselor software destinate a funcționa în timp real implică mai mulți paşi, descrişi în cele ce urmează.

5.1 Stabilirea arhitecturii software

Acesta este prima etapă a proiectării produselor SW care funcționează în R-T. În acest pas, echipa de programatori trebuie să înțeleagă sistemul pentru care se doreşte proiectarea produsului SW. De asemenea, proiectanii trec în revistă arhitectură HW dorită şi implementează o arhitectură SW foarte de baza, care va fi apoi dezvoltată.

Cazurile de utilizare sunt de asemenea folosite în acesta etapă pentru a analiza sistemul. Ele ajuta la înțelegerea interacțiunii dintre sistem şi utilizatorii săi. De exemplu, cazurile de utilizare pentru o centrală telefonică presupune interacțiunea dintre schimburile de telefoane, abonații şi operatorii telefonici.

5.2 Co-Design

Odată ce arhitectură SW a fost definite, echipele HW şi SW trebuie să lucreze împreună pentru a asocial functionatitatea SW modulelor HW. Software-ul trebuie controlat pentru oeficientă partiționare între procesoare diferite şi celelalte resurse HW, cu următoarele considerații cheie:

- Funcționalitatea SW ar trebui partajată în aşa fel încât procesoarele şi conexiunile din sistem să nu se suprapună atunci când sistemul operează la capacitate maximă. Acest lucruimplică simularea sistemului cu arhitectură SW şi HW propusă.

- Sistemul ar trebui proiectat pentru o viitoare extindere (îmbunătățire) prin considerarea unei arhitecturi scalabile. Sistemul nu va fi scalat bine dacă anumite module HW sau SW devin prea aglomerate când capacitatea sistemului creste.

- Modulele SW care interacționează direct unele cu altele ar trebui plasate pe acelaşi procesor, ceea ce va reduce eventualele întârzierile din sistem şi va creste performantele.

Această etapă se numeşte adeseori Co-Design, din moment ce echipele HW şi SW lucrează împreună pentru a define arhitectură finală a sistemului, ceea ce poate fi un proces iterative.

5.3 Definirea subsistemelor SW

Se începe cu determinarea altor parametrii sau funcții de care sistemul va avea nevoie sau trebuie să le îndeplinească.

Următorul pas este gruparea funcțiilor diferite pe tipul de sarcini pe care le executa. Astfel se identifică mai multe subsisteme, prin atribuirea unui subsistem SW unui astfel de grup de funcții.

În continuare, se identică sarcinile prin care se vor implementă aceste caracteristici SW, şi se defineşte clar funcția fiecărui subsistem.

În ultimul etapă, pentru fiecare subsistem, se vor clasifică şi grupă toate caracteristicile apropiate şi li se va asocia anumite task-uri.

5.4 Proiectarea caracteristicilor produsului

Un sistem tipic care funcționează în timp real este compus din diferite sarcini distribuite procesoarelor component şi toată comunicația dintre procesoare are loc în principal prin transferul de mesaje. Proiectarea caracteristicilor defineşte caracteristicile SW în termeni de interactiilor mesajelor dintre task-uri. Acesta etapă presupune mai mulți paşi:

- Specificarea interacțiunilor mesajelor dinte sarcinile diferite din sistem;
- Identificarea activitatiilor care vor controlată aceste caracteristici. Aceste sarcini de control vor urmări progresul caracteristicilor. În general, acest lucru este realizat prin utilizarea timer-elor.
- Interfețele dintre mesaje sunt stabilite în detaliu, identificandu-se toate câmpurile şi valorile posibile care pot influența comunicația.

Instrucțiuni pentru proiectarea caracteristicilor produselor SW

- Păstrarea unui design simplu şi definirea clară a arhitecturii sistemului;
- Implicarea a cât mai puține sarcini pentru fiecare ;
- Împărțirea caractersticilor mari şi complexe în unele mai mici şi simple ;
- Clasificarea posibilelor interacțiuni dintre mesaje;
- Furnizarea unei proiectări clare şi complete a fiecărei interfețe dintre mesaje;
- Întotdeauna trebuie asigurta posibilitatea de recuperare a datelor anterioare şi revenirea la starea inițial;
- Pentru evitarea supraîncărcării legăturilor dintre mesaje, se aleg alternative de proiectare care să include mai puține schimburi între mesaje.

5.5 Proiectarea sarcinilor

Proiectarea unei sarcini necesită că toate interfețele pe care trebuie să le suporte ai trebui să fie bine definite. Trebuie asigurat faptul că toți parametrii mesajelor şi valorile timerelor să fie finalizate.

Selectarea tipului de sarcina

Odată ce interfețele externe sunt isolate, se selectează tipul/tipurile de sarcini care sunt şele mai apropiate pentru a controla interfețele:

- **Maşini cu o singură stare**: funcționalitatea taskurilor poate fi implementată întrèo maşină cu o singură stare.
- **Maşini cu stări multiple**: sarcina controlează o maşină cu stări multiple. Astfel de sarcini includ de obicei un distribuitor de probleme care să atribuie mesajele recepționate celei mai apropiate stări ale maşinii. În plus, ele crează sau şterg obiectele maşinii de stare atunci când e nevoie.
- **Sarcini multiple**: aceste tipuri de sarcini sunt similar cu taskurile maşinilor cu stări multiple discutate anterior. Diferența majoră dintre cele două este faptul că acum sarcinacontrolează mai multe

acțiuni. Fiecare task rezolvat reprezintă o maşină cu o singură stare. Sarcina manager este responsabilă şi de crearea şi ştergerea taskurilor maşină cu osingură stare.

- **Sarcina complexă**: acest tip de task este necesară în scenario foarte complexe. În acest caz, sarcina manager controlează alte sarcini care ar putea răspunde de maşini cu stări multiple.

Selectarea unui design pentru maşină de stare

După alegerea tipurilor de tarcini, proiectantul trebuie să ia în considerare divizarea interfețelor dintre mesajelor care răspund de o secvență de tranziții de stări. Două astfel de maşini pot fi posibile:

- **Maşini cu stări plate:** acestea sunt cele mai frecvente tipuri utilizate . acest tip de divizare a stărilor nu este scalat bine cu creşterea complexității. Pentru un sistem complex, această tehnică va genera i explozie de stare, cu o sarcina care necesită sute de stări.

- **Maşini cu stări ierarhice:** în acest caz stările sunt văzute că o ierarhie. Numărul total de stări este acelaşi, cu diferența că anumite stări sunt generate din alte stări. Numărul total de tratări ale mesajelor pentru fiecare stare este redus drastic, din moment ce toate mesajele care au un intermediary în toate stările originale va trata doar starea originală.

6. Probleme întâmpinate la proiectarea produselor SW

(Ana – Maria Radu)

Proiectarea sistemelor care funcționează în timp real este o adevărată provocare. Principal problemă este faptul că aceste sisteme trebuie să interacționeze cu entități din lumeareală. Aceste interacțiuni pot ajunge să fie foarte complewe. Un sistem tipic care funcționează în R-T poate interacționă cu sute de astfel de entități în acelaşi timp. De exemplu, un sistem de telefonie trebuie să dirijeze zeci sau chiar sute de telefonane de la diferiți utilizatori, sistemul trebuind să se conecteze diferit la fiecare telefon în parte. Înplus, şirul exact de evnimente în apelul telefonic poate varia foarte mult. În continuare sunt prezentate cele mai comune probleme întâmpinate de proiectanți :

6.1 Răspunsul în timp real

Sistemele care funcționează în timp real trebuie să răspundă interacțiunilor externe într-un anumit timp predefinit de timp. Completarea cu success a unei operațiuni depinde de corectitudinea şi timpul execuției fiecărui task a sistemului. Aşadar, proiectarea produselor SW şi HW ale unui sistem trebuie să fie în concordanță cu cerințele pentru răspunsul întimp real. De exemplus, o central telefonică trebuie să manevreze sute de apeluri de la apelanți într-un număr limitat de secunde (timp foarte scurt). Pentru a îndeplini aceste cerințe, mecanismul de detecție al tonului de ocupat cât şi software-ul

pentru comunicație implicate trebuie să funcționeze într-un anumit interval de timp. Astfel, central telefonie trebuie să îndeplinească cerințele în ceea ce priveşte timpul de răspuns în orice moment.

Proiectanții acestor tipuri de produse SW trebuie să se concentreze încă de la început pe cerințele răspunsului în timp real. În timpul perioadei de proiectare, inginerii hardwareşi software lucrează împreună pentru a găsi arhitectură sistemului cea mai potrivită care să răspundă cerințelor. Din acest punct de vedere, principalele întrebări pe care trebuiesă şi le pună proiectanții produselor SW sunt:

- **Arhitectură este stabilă?** Dacă sistemul de comunicație implică prea multe noduri, este foarte posibil că sistemul să nu poate reacționă în timp real din cauza congestiilor. Aşadar, o arhitectură mai simplă este de preferat.

- **Viteză de conexiune este potrivită?** În general, încărcarea unei conexiuni mai mult de 40-50% este o idee rea. Încărcarea unei legături cauzează cozi la transferul datelor prin anumite noduri ale rețelei şi congestii, provocând astfel întârzieri diferite la comunicare.

- **Elementele de procesare sunt sufficient de puternice?** Un processor cu o foarte mare utilizare poate conduce la un comportament în timp real neprevizibil. De asemenea, este posibilcă sarcinile ce cea mai mare prioritate să ocupe prea mult din timpul alocat de processor sarcinilor cu o prioritate redusă. Acest lucru poate cauza eşuări ale sarcinilor cu prioritate scăzută.

- **Sistemul de operare este potrivit?** Taskurile care sunt implicare în evenimentele critice de procesare în timp real au o prioritate ridicată. Atunci când alegem sistemul de operare, latent întreruperilor şi programarea variantei trebuie să fie corespunzătoare cerințelor sistemului.

    - Programarea variantei se referă la predictibilitatea timpului alocat fiecărei sarcini. De exemplu, o centrală telefonică trebuie să asigure disponibilitatea liniilor telefonice lamai puțin 500 ms. În mod obişnuit, acest lucru ar implică programarea a trei până la cinci sarcini în timpul considerat. Majoritatea sistemelor de operate pot îndeplini aceastăcondiție dacă este luat în calcul şi media timpului de ton de apel. Dar sistemele de operare generale au o deviere standard mult mai mare în ceea ce priveşte această problemă.
    - Latentă întreruperilor se referă la întârzierea cu care sistemul de operare manevrează întreruperile şi sarcinile programate pentru a răspunde unei întreruperi. Amintim faptul căsistemele de operare care funcționează în timp real ar trebui să aibă o latentă a întreruperilor mult mai mică.

    Timpul de raspuns este un important factor in toate sistemele dedicate dar in unele cazuri, raspunsul foarte repede nu e necesar. Un sistem de timp real poate fi vazut ca un sistem stimul/raspuns. Avand un anumit stimut la input, sistemul trebuie sa produca un raspuns corespunzator. Comportamentul

sistemului de timp real va fi evaluat deci luand in considerare stimulii primiti, raspunsurile corespunzatoare emise de sistem si durata dupa care trebuie sa se produca acest raspuns. Stimulii se impart in doua categorii:

- Stimul periodic – acestea au loc la interval predictibile de timp. De exemplu, sistemul poate examina un sensoz la fiecare 50 milisecunde si apoi sa ia o masura(raspuns) depizand de valoarea senzorului(stimului).
- Stimul aperiodic – acesta nu mai e predictibil. Acestia sunt folositi uzual in mecanismele de intrerupere. Un exemplu a cestui stimul, cand o intrerupere ne indica ca un transfer I/O este complet sic a datele sunt disponibile intr-un buffer.

In sistemele de timp real, stimulii periodici sunt generati de senzorii asociati sistemului, oferind informatii despre starea mediului sistemului. Raspunsurile sunt trimise catre un set de actuatori, care vor controla o parte din echipamentele, influentand mediul sistemului. Stimulii aperiodici pot fi generati de senzori sau de actuatori, indicand adesea conditii extraordinare, precum o defectare hardware, care trebuie sa fie rezolvata de sistem.

Evenimentele (sau stimulii) ar trebui sa fie centrale proceseului de design software, si nu obiectele sau functiile. Exista cateva stadii suprapuse pentru procesul de design:
1. identificarea stimulilor care trebuie procesati si raspunsurile asociate
2. pentru fiecare stimul si raspuns asociat, identificarea constangerilor de timp care se aplica
3. alegerea unei platforme de executie pentru sistem: hardware-ul si sistemul de operare in timp real care trebuie folosit. Factorii care influenteaza aceste alegeri include constrangerile de timp asupra sistemului, limitarile de putere disponibila, experienta echipei de dezvoltare si pretul tinta a sistemului livrat
4. agregarea procesarii stimulilor si raspunsurilor intr-un numar de procese concurent. In designul sistemelor in timp real, trebuie asociat un proces cu fiecare clasa de stimuli si raspunsuri
5. pentru fiecare stimul si raspuns, trebuie proiectati algoritmi care sa execute calculele necesare. Proiectarea acestor algoritmi trebuie sa fie facuta relativ devreme in procesul de design.
6. Proiectarea unui sistem de planificare care va asigura ca procesele se pornesc la timpul potrivit si sunt executate in timpul necesar.

6.2 Recuperarea după defectări

Sistemele care funcționează în R-T trebuie să fie fiabile în evetualitatea eşuărilor, atât externe cât şi interne. Î n continuare sunt prezentare modurile de tratare a astfel de erori.

Defecțiuni interne

Pot fi cauzate de eşuări HW sau SW ale sistemului. Cele mai tipice defecțiuni interne sunt:

- **Apariția unui bug în timpul unei sarcini**: spre deosebire de aplicațiile desktop, aplicațiile care funcționează în R-T nu generează o casetă de dialog atunci când sistemulîntâmpină o eroare. Proiectarea taskurilor trebuie făcută astfel încât să fie protejate în eventualitatea unor defecțiuni. Acest lucru devine şi mai important pentru sistemele întimp real deoarece pot există o mulțime de scenării şi evenimente posibile. Este foarte posibil că nu toate scenăriile să fie testate în laborator în timpul proiectării produsului SW. Aşadar, este important să existe posibilitatea de salvare periodică a datelor şi task-urilor, în eventualitatea apariției unor erori. În plus, anumite condiții de eroare software poate conduce la generarea unei execeptii a procesorului. În astfel de cazuri, trebuie să fie posibilă şi restaurarea task-urilor înainte de excepții şi a datelor anterior salvate.

- **Restartarea procesorului**: Multe sisteme R-T sunt compuse noduri multiple. Nu este posibilă defactarea întregului sistem dacă un sisngur nod este defect, attfel încât laproiectarea prodului SW trebuie tratate defectările independente pentru fiecare nod. Acest lucru implică două activități:
    1. Tratarea defectării procesorului: când un processor se defectează, celelalte procesoare trebuie să fie notificate de această defecțiune. Aceste procesoare vor anulă orice interacțiuni cu procesorul defect.
    2. Recuperarea datelor unui processor defect: atunci când un processor, inițial defect, este înlocuit, el trebuie să recupereze datele pierdute în urmă defectării de la celelalte procesoare din sistem. Întotdeauna există şansă unei neconcordante între procesoare diferile din sistem, caz în care sistemul trebuie să găsească modalități de rezolvare a acestor diferențe.
- **Defectarea plăcutei de baza** : sistemele care funcționează în R-T trebuie proiectate astfel încât să îşi revină după o defecțiune HW. Sistemul ar trebui să poată detectă şirecuperă după defecțiuni ale placulelor HW. Atunci când apare o astfel de defecțiune, sistemul notifică operatorul de problemă întâmpinată, iar când această este înlocuită saureparată, el trebuie să revină la starea inițială.
- **Eroarea de conexiune**: majoritatea comunicațiilor într-un sistem care funcționează în R-T au loc prin intermediul unor conexiuni între diferite procesoare din sistem. Şi în acest caz, sistemul trebuie să izoleze conexiunea care nu funcționeze şi să redirecționeze mesajele, astfel încât legătură problemă să nu afecteze comunicarea.

Defecțiuni externe

Sisteme R-T trebuie sa functioneze in lumea reala. Astfel, ele trebuie sa isi revina dupa anumi defectari de natura externa. Un sistem poate fi supun unor defectari, precum :

- **Compotament invalid ale entitatiilor externe** : atunci cand un sistem R-T interactioneaza cu entitati externe, el ar trebui sa poata sa manevreze toate conditiile posibile de defectari ale acestor entitati sau ale comportamente nepotrivita ale utilizatorilor.

- **Eroare intre conexiuni** : de multe ori sistemul este distribuit in mai multe locatii, astfel incat alte legaturi externe se pot conecta la aceste locatii. A manevra aceste conditii este echivalent cu a face fata erorile conexiunilor interne. Diferenta majora a acestor erori este durata extinsa si faptul ca in general nu este posibila redirectionarea mesajelor.

6.3 Lucrul cu arhitecturi distribuite

Sisteme R-T trebuie să funcționeze în lumea reală. Astfel, ele trebuie să își revină după anumi defectări de natură externă. Un sistem poate fi supun unor defectări, precum :

- **Compotament invalid ale entitatiilor externe** : atunci când un sistem R-T interacționează cu entități externe, el ar trebui să poată să manevreze toate condițiile posibile de defectări ale acestor entități sau ale comportamente nepotrivită ale utilizatorilor.

- **Eroare între conexiuni** : de multe ori sistemul este distribuit în mai multe locații, astfel încât alte legături externe se pot conectă la aceste locații. A manevră aceste condiții este echivalent cu a face fața erorile conexiunilor interne. Diferența majoră a acestor erori este durată extinsă și faptul că în general nu este posibilă redirecționarea mesajelor.

6.4 Comunicarea asincronă

Procedurile de apelare de la distanță (RPC – remote procedure calls) sunt folosite de către programatori pentru simplificarea design-ului produselor SW. RPC permit programatoruluisă apeleze funcții ale unei mașini mai îndepărtate folosind aceasi semantică că și prodesurile de apelare locale. Astfel, RPC simplifică foarte mult proiectarea produselor SW șiajuta la dezvoltarea sistemelor convenționale. Cu toate acestea, ele sunt foarte puțin folosite la sistemele care funcționează în timp real. Pricipalul motiv pentru acet lucru este faptul că în general, comunicarea în lumea reală este de natură asincronă (de exemplu, mesajele în general nu sunt într-o anumită ordine secvențială, ceea ce s-ar potrivi folosind RPC).

Așadar, majoritatea sistemelor în R-T prezintă mașini de stare bazate pe arhitectură unde mesaje multiple pot fi recepționate într-o singură stare. Următoarea stare este determinate de conținutul mesajelor primate. Mașină de stare furnizează un mecanish foarte flexibil de tratarea interctiunilor mesajelor asincrone.

7. Analiza performanțelor în timp real

(Madalin Nica)

Analiză performantelor în timp real este o component esențială a sistemelor ce operează în timp real şi este folosită pentru a estimă dacă un sistem se va comportă conform aşteptărilor [14] .

7.1 Estimarea timpului de execuție

Pentru a estima performanța unui sistem concurent trebuie mai întâi să putem estimă perfomanta unei singure sarcini secvențiale. Estimarea timpului de execuție limitează rularea unui segmenent cu un singur fir de execuție când acesta va rulă pe un processor dedicat. Problemă estimării timpilor de execuție este compusă din două sub-probleme:

- Identificarea ordinea corectă a instrucțiunilor prin cod ;
- Determinarea timpului de exectuie a fiecare instrucțiuni din cod.

Estimarea corectă a timpilor de exectuie ce limitează folosing tehnici analitice este grea de realizat dintr-un număr de motive. Primul, în general, este greu de decis ordinea tuturor instrucțiunilor ce se executa într-un program. În al doilea rând timpul de exectuie a fiecărei instrucțiuni este dependent. În ultimul rând caracteristicile arhitecturale ale procesoarelor moderne, cum ar fi cache şi pipeline îngreunează determinarea cu acuratețe a timpilor de exectuie datorită efectelor lor globale. În practică, datorită limitării tehnicilor analititce, trebuie să ne bazăm pe măsurători şi testări de lungă durată pentru a determină cu acuratețe limitarea impusă de timpul de execuție estimate.

7.2 Programarea în timp real

Analizarea timpilor de execuție pentru o sarcina individuală este un pas important asta nu înseamnă că este şi suficientă. Când se folosesc mai multe fire de execuție trebuie săcondiseram şi programarea sarcinilor în processor. Programarea în timp real pentru un singur processor consistă în alegerea ordinii de exectuie a unui set de sarcini (fire de execuție concurente) cu caracteristici cunoscute (ex : periodicitate şi timpi de execuție).

Cea mai simplă abordarea a programării în timp real este construirea unui orar off-line. În timpul execuției, sarcinile sunt executate conform orarului prestabilit. Acest tip de programare, statică, este aplicabil în mod particular atunci când majoritatea activităților sunt bazate pe execuția periodică. Un orar cyclic poate fi creat pentru un set de sarcini reale ciclice. Abordarea ciclică tradițională este un exemplu de astfel de programare.

Programarea statică ciclică este inadecvată când evenimentele pot apărea aperiodic. Această abordare poate fi folosită, ea însă poate duce la utilizarea ineficientă a procesorului.În schimb, o programare în timp real este folosită în acel caz. Cea mai des întâlnit instrument de programare în timp real este abordarea priorităților preventive, în cadrul căreia sarcinile sunt rulate în timp real conform priorităților şi tot timpul cele cu prioritatea cea mai mare sunt rulate.

O analiză off-line (sau simularea) trebuie să complementeze un orar în timp real pentru a asigură faptul că cerințele de rulare în timp real sunt îndeplinite. Analiză programării este folosită pentru

a determină dacă un sistem dat va face fața cerințelor de execuție la timp în tim real. Există o teorie bine dezvoltată, bazată pe programarea priorităților preventive şi analiză determinist ace poate fi folosită pentru a previziona timpii ce mai mari de răspuns la un eveniment, având cunoscute rață maximă de sosire, timpul maxim de execuție şi o prioritizare a fiecărei sarcini.

Sincronizarea între sarcini complică analiză programării. Când exclurerea mutual este folosite între sarcini inversia priorităților poate apărea.O sarcina de prioritate redusă poateblocă execuția unei sarcini de prioritate mare când amandoia împart aceeaşi secțiune critică. Mai rău de atât, aceastra inversare a priorității poate fi întreruptă când un număr arbitrar de sarcini cu prioritate medie o precedeaza pe cea de prioritate mică. Din fericire există un număr de protocoale pentru moştenirea priorității care ne permit să inversămschimbul de priorități şi să putem analiză programarea. Ideea de baza în spatele acestor protocoale este să forțeze execuția sarcinei de prioritate mică într-o secțiune criticăînaintea sarcinilor cu prioritate medie care urmează după ea.

Mai multe forme generale ale sincronizării duc la îngreunarea analizei programării, uneori chiar la imposibilitatea realizării ei. Însă, o sincronizare precedentă poate fi analizatăfolosind abordarea analizei deterministe. Sincronizarea precedentă este folosită în comunicația sincronă.

Deşi analiză programării este cum o tehnică bine stabilită, ea este bazată pe câteva presupuneri cum ar fi timpi de execuție max prosti, concurență simplă, sincronizare, modele de comunicații, etc. Deşi se folosesc o varitate de analize a performantelor atunci când aceste presupuneri nu sunt îndeplinite (ex verificarea modelelor, analiză performantestochastice, simulări) ele încă reprezintă o problemă dificilă. .

8. Tehnologii în timp real, instrumente şi standarde

(Dan Arama)

O parte importantă în dezvoltarea soft-ului în timp real este utilizarea instrumentelor şi tehnologiilor ce susțin procesul de dezvoltare. Î n acest capitol sunt prezentate sumar tehnologiile, instrumentele şi standardele disponibile programatorilor de produse SW pentru aplicațiile ce funcționează în timp real.

8.1 Sisteme de operare în timp real

Dacă tradițional multe aplicații în timp real au fost construite folosind un sistem de operare, utilizarea sistemelor de operare în timp ce structurează mai bine soft-ul este indispensabilă atunci când aplicația utilizează activități concurente multiple.

Un sistem de operare în timp real oferă abstractizări pentru activitățile concurente, comunicarea şi sincronizarea mecanismelor dintre activitățile concurente. Acest lucru simplfică scrierea produlului SW. Majoritatea sistemelor de operare în timp real oferă programarea preventivă - prioritară a sarcinilor. Managementul priorităților pentru activitățile concurente ce satisfac cerințele de timp este lăsat în sarcina aplicației.

Î n mod tradițional, sistemele de operare în timp real dispun de propriile API. Apariția standardului POSIX a ajutat la standardizarea câtorva distribuitori de sisteme de operare ce dispun de suport POSIX pentru API şi în mod special cei care utilizează extensii în timp real.

8.2 Instrumente de design şi dezvoltare

Tradițional instrumentele pentru design-ul şi dezvoltarea soft-ul în timp real nu sunt foarte sofisticate. Aceasta se întâmplă datorită naturii intrinseci a soft-ului în timp real şi embedded. Î nsă, industria a început să investească, într-un final, în instrumentele de dezvoltare ceea ce este un semn bun. De exemplu, mulți furnizori oferă acum integrat un mediu de dezvoltare cu instrumente pentru editare, compilare, profilare, monitorizare, debugging, managementul configurațiilor, etc. Multe dintre aceste instrumente sunt cross-hosted, de exemplu instrumentele de dezvoltare rulează pe platforme de găzduire (de obicei Windows NT sau Unix), dar suportul comunicării cu soft-ul în timp real rulează pe maşina locală.

Unele instrumente merg mai departe cu acest mediu de programare, ele suportând modele bazate pe design în timp real. Aceste instrumente oferă abilitatea de a dezvolta produse SW folosind modele high-level şi oferă suport pe parcursul ciclului de viată de la analiza cerințelor de design până la implementare.

8.3 Instrumente de analiză a performanței

Instrumentele de analiză a performanței ajută în analiza proprietăților soft-ului în timp real. Î n mod ideal, asemenea instrumente oferă estimări timpurii asupra fezabilității designurilor utilizabile bazate pe estimarea costului de rulare şi apoi să suporte analiza de la etapa de design până la implementare, unde costurile reale pot fi măsurate, şi analiza performanței să fie validate pentru rularea în timp real. Din păcate aceasta este o zonă relativ slabă în ceea ce priveşte dezvoltarea instrumentelor, deşi câteva instrumente au ajuns să suporte analiza programării şi simulării modelelor pentru proprietățiile de rulare in timp.

9. Viitorul

(Dan Arama)

În această lucrare am analizat problema proiectării software în timp real, concentrându-ne asupra 2 aspecte cheie, mai exact concurența şi temporalitatea. De asemenea, am prezentat diferite stiluri de proiectare. În încheiere, am prezentat câteva dintre tehnologiile şi instrumentele disponibile proiectanților software în timp real.

Este încurajator să observăm cum proiectarea software în timp real depăşindu-şi faza injust conservative şi adoptând, în sfârşit, instrumente şi tehnologii moderne. Sunt, în mod evident, multe aspecte care pot fi îmbunatațite. Unele dintre cele mai mari deficiențe constau în modul în care se poate trata problema temporalității. De exemplu, uneltele de analiză a performanței sunt relative primitive, nefiind in concordanță cu mediile de proiectare şi modelare. Este, de asemenea, un conflict între modelele de design generalizate şi capacitatea de a analiza temporalitatea lor. Mare parte dintre sisteme este soft în timp real, existând o

nevoie reală de instrumente care să susțină proiectarea de sisteme flexibile în timp real a căror performanțe suferă o degradare în maniere previzibile şi controlate în situatiile de suprasolicitare.

O altă zonă în care preconizăm multe provocări este în metodologia cercetăriilor în proiectare. În dezvoltarea de software complex în timp real, mai multe posibilități de proiectare îi sunt disponibile programatorului. În absența unor îndrumări şi euristică, astfel de alegeri sunt făcute într-o manieră ad-hoc, şi pot duce uşor la o proiectare ineficientă. Tendință spre abstractizări şi modele de proiectare de nivel înalt şi care sunt mai apropiate de domeniul problemei exacerbează problema, deoarece duce la mai multe alegeri când se face trecerea de la abstractizările caracteristice proiectării la abstractizările caracteristice implementării. De exemplu, într-un stil de proiectare orientat pe obiecte, alegerile trebuie făcute ținând cont de modul în care modelul este asociat cu firul sistemului de operare, cum sunt desemnate prioritățile, etc.

Pe măsură ce societatea evoluează spre o dependență simbiotică cu sistemele computerizate, o parte mereu crescândă va aparține de sisteme din categoria largă a sistemelor în timp real. Ceea ce presupune că metodele şi tehnologiile create pentru software în timp real tradițional, incluzând, în particular, tehnici pentru toleranța greşelilor şi previzibilitate, vor deveni parte a pachetului standard de cunostiințe a marii majorități a inginerilor şi proiectanților software. Pe deasupra, vor fi susținuți de pachete de instrumente din ce în ce mai sofisticate, care vor duce la creşterea nivelului de automatizare a proiectării software în timp real.

Bibliografie

[1] Jackson, “Principles of Program Design,” *Academic Press, New York,* 1975.

[2] K. Orr., “Structured Systems Development,” *Yourdon Press, New York,* 1977.

[3] G. Myers, “Composite/Structured Design,” *Van Nostrand Reinhold,* 1976.

[4] E. a. C. Yourdon, “Structured Design,” *Yourdon Press, New York,* vol. 2nd Edition, 1978.

[5] D. Marco, “Structured Analysis and System Specification,” *Yourdon Press, New York,* 1978.

[6] S. a. Gane, “Structured Systems Analysis : Tools and Techniques,” *Prentice-Hall, Englewood Cliffs,* 1979.

[7] Hamilton and Zeldin, “Higher Order Software - A Methodology for Defining Software,” *IEEE. Trans. Softw. Eng.,* March 1976.

[8] H. Simpson and M. Jackson, “Process Sychnonization in Mascot,” *Comput. J.,* 1979.

http://www.scrigroup.com/calculatoare/Proiectarea-programelor-pentru42622.php

[9] D. Parnas, “On the criteria to be used in decomposing systems into modules,” *Commun. ACM,* pp. 1053-1058, 1972.

[10] G. Booch, “Software Engeneering,” *Menlo Park, Calif.,* 1983.

[11] E. W. Dijkstra and F. Genuys, Co-operating sequencial processed in programming languages, New York: Ed. Academic Press, 1968.

[12] P. Brinch Hansen, “Concurent programming concepts,” *Comput. Surv.,* pp. 223-245, 1973.

[13] P. Jalote, “Fault Tolerance in Distributed Systems,” *Prentice-Hall,* 1994.

[14] M. Klein, A Practicioner's Handbook for Real-Time Analysis : Guide to Rate Monotonic Analysis for Real-Time Systems, Kluwer Academic Publishers, 1993.