Intelligent System for Real Time Acquisition of the Distributed Data

Nicoleta Cristina GĂITAN, Vasile Gheorghiță GĂITAN, Ştefan Gheorghe PENTIUC

***Abstract*—For real time applications, the specific time requirements represent the main constraints, and their controlling is an essential factor in order to establish the quality of the accomplished services. The time constraints might be found within more application areas, such as the automatisation of industrial equipments, the embedded systems, the control of vehicles, monitoring the nuclear devices, supervising the scientific experiments, robotics, conditioning of audio and video multimedia data flows, monitoring the surgical interventions, as well as supervising the stock exchange operations. Within this paper, the hardware and software architecture of an intelligent system for real time acquisitions of the distributed data is presented. This intelligent system is represented by means of an embedded device, performed with a STR710FZ2 microcontroller and a RTX real time kernel.**

***Index Terms*—embedded device, real-time kernel, intelligent system, microcontroller**

I. INTRODUCERE

Prezentul articol prezintă arhitectura hardware şi software a unui sistem inteligent de achiziții în timp real a datelor distribuite.

Acest sistem inteligent se prezintă sub forma unui dispozitiv *embedded* realizat cu ajutorul unui microcontroler STR710FZ2 şi a nucleului de timp real RTX**.**

Un sistem de timp real este un sistem al cărui corectitudine nu depinde doar de rezultatul logic al unui calcul, dar şi de momentul la care rezultatul a fost produs. O definiție echivalentă se centrează pe comportamentul sistemului observabil din exterior. Acest lucru poate fi descris de către noțiunea de service: Service-ul (-urile) furnizate de către un sistem este (sunt) comportamentul său observat de către utilizator(-i)[[7]] [[8]], prin urmare de către mediul său. Bazându-ne pe observația anterioară, service-ul de la un sistem de timp real trebuie să satisfacă cerințele temporale cât şi cele funcționale (logice); altfel sistemul trebuie să fie luat în considerare ca fiind „eronat". Acest lucru motivează următoarele definiții[[1]]:

1. Def: Service-ul de timp real este un service care este necesar ca să fie furnizat în intervale de timp dictate de mediu.
2. Def: Un sistem de timp real este un sistem care furnizează cel puțin un service de timp real. De notat că această definiție subliniază de asemenea faptul că părți (service-uri) diferite ale aceluiaşi sistem pot fi subiect pentru necesități temporale, diferite sau chiar fără.

Aplicațiile complexe necesită mai multe resurse de calcul ceea ce este în mod frecvent mai uşor de obținut prin utilizarea a câtorva procesoare. Aplicațiile critice oferă o creştere pentru necesitățile mult mai stricte care pot fi îndeplinite doar prin rezolvarea corespunzătoare a mecanismelor cu toleranță la defect (eroare) [3]. Toleranța la defect (eroare) necesită redundanță, ce poate fi realizată prin utilizarea de procesoare suplimentare.

Un sistem în timp real prezintă *constrângeri de timp severe*, atunci când o eroare de timp (neîndeplinirea unui termen limită, livrarea unui mesaj prea târziu, eşantionarea datelor neregulat, dispersia prea mare a datelor presupuse a fi colectate simultan) poate cauza dezastre umane, economice sau ecologice. Un sistem în timp real prezintă *constrângeri de timp uşoare* atunci când erorile de timp pot fi tratate cu o anumită extensie. Un sistem computerizat în timp real este un sistem al cărui comportament este stabilit de dinamica unei aplicații. Astfel, o aplicație în timp real constă din două părți conectate: sistemul computerizat de control în timp real şi procesul controlat.

La sistemele în timp real, cuvântul *sarcină de lucru (task)* este cel mai adesea utilizat ca unitate pentru reprezentarea activităților concurente ale arhitecturii logice. Paralelismul fizic din arhitectura hardware şi paralelismul logic din cerințele aplicației reprezintă, de cele mai multe ori, baza pentru împărțirea unei aplicații în sarcini concurente.

II. ARHITECTURA HARDWARE A UNUI MASTER INTELIGENT PENTRU ACHIZIȚIA DATELOR DISTRIBUITE

Arhitectura hardware a unui master inteligent oferă resursele hardware necesare pentru implementarea aplicației în timp real denumită Master Inteligent. Microcontrolerul trebuie să asigure o putere de calcul sporită la un preț de cost redus. La această oră arhitecturile ARM7 TDMI şi Cortex M3 oferă resurse suficiente pentru astfel de aplicații. Memoria estimată este de 128 kocteți de memorie flash şi, datorită numărului mare de buffere de comunicație, minim 64 kocteți SRAM.

Pentru a realiza istorice pentru intrări ale dicționarului de obiecte şi pentru configurarea protocoalelor de comunicație şi a ciclului de achiziție, sistemul a fost prevăzut cu o memorie nevolatilă de tip SD sau MMC.

Pentru trasarea timpului se va alege un microcontroler cu

Cristina Găitan – Universitatea Ştefan Cel Mare Suceava, str.Universitatii nr.13, RO-720229 Suceava (e-mail: cristinag@eed.usv.ro)

Vasile Găitan– Universitatea Ştefan Cel Mare Suceava, str.Universitatii nr.13, RO-720229 Suceava (e-mail: gaitan@eed.usv.ro)

Ştefan Gheorghe Pentiuc – Universitatea Ştefan Cel Mare Suceava, str.Universitatii nr.13, RO-720229 Suceava (e-mail: pentiuc@eed.usv.ro)

ceas încorporat. Pentru comunicația cu PC-ul au fost prevăzute 3 posibilități și anume:

1. comunicația utilizând portul serial și standardul liniei RS232;
2. comunicația utilizând portul USB;
3. comunicația utilizând o conexiune Ethernet de tip 10/100 Mb/sec.

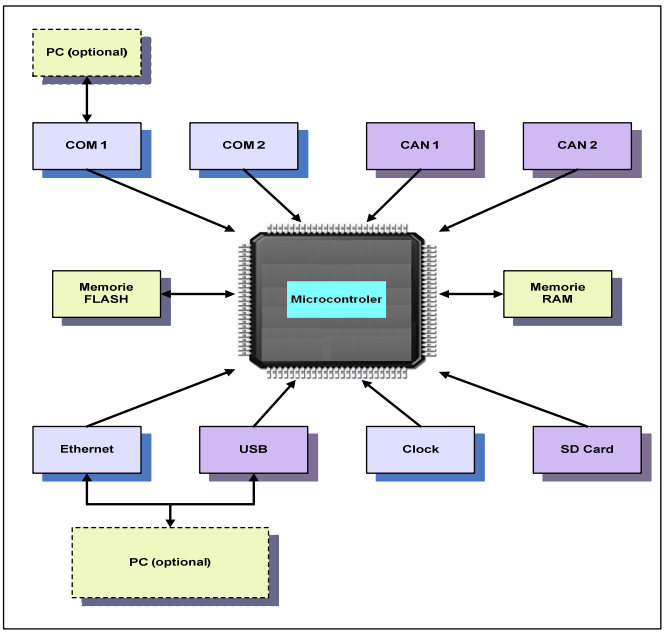

Fig. 1. Interfațarea sistemelor digitale la mediul înconjurător.

Pentru implementarea protocoalelor impuse de cerințele definite, arhitectura a fost prevăzută cu:

- porturi seriale pentru protocoalele ASCII, M-Bus, Modbus RTU/ASCII și comunicație prin modem radio (se vor folosi standardele de linie RS232 și RS485);
- porturi Ethernet care pot asigura protocolul de tip Modbus TCP/IP;
- porturile CAN care permit implementarea uneia sau a mai multor rețele bazate pe protocolul CANopen.

Este de preferat, din motive de fiabilitate și spațiu pe cablaj, ca pe cipul microcontrolerului să fie integrate cât mai multe dintre resursele prezentate. Acesta va fi un criteriu important de selecție a microcontrolerului.

Provocarea pentru standardele de timp real constă în a alege între nucleele de timp real care sunt standardizate prin adoptarea interfeței standard Unix și nucleele de timp real non-Unix, modificate pentru a oferii facilități specifice timpului real.

Un set de interfețe de programare a aplicațiilor (API), care extind interfața Unix către timp real, au fost propuse ca standardul Posix 1003.1b. Aceste interfețe, care permit portabilitatea aplicațiilor cu cerințe de timp real, sunt:

- funcțiile interfeței timer-ului pentru a seta și a citi timerele interne de mare rezoluție;
- funcțiile de planificare care permit preluarea și setarea parametrilor de planificare. Sunt definite trei politici: SCHED_FIFO, o planificare *preemptive*, bazată pe priorități, SCHED_RR, o planificare *preemptive*, bazată pe priorități cu cuantă (round-robin) și SCHED_OTHER, o planificare definită de către implementare.
- funcțiile fișierelor care permit crearea și accesul la fișiere cu performanțe deterministe.
- primitive de sincronizare eficiente precum semafoarele și facilități pentru transmiterea sincronă și asincronă de mesaje.
- funcțiile pentru notificarea asincronă a evenimentelor și a semnalelor de timp real plasate în cozi.
- funcțiile pentru blocarea memoriei unui proces și facilitare a mapării memoriei partajate.
- funcțiile eficiente care realizează operațiile I/O sincrone și asincrone.

III. SISTEMUL DE OPERARE ÎN TIMP REAL

Un sistem de operare în timp real trebuie să furnizeze facilități pentru îndeplinirea a trei cerințe importante ale aplicațiilor în timp real:

- garantarea unui răspuns de la sistemul de calcul;
- promptitudinea răspunsului, odată ce acesta a fost decis;
- siguranța codului aplicației.

Un sistem de operare *în timp real* trebuie să fie capabil să ia în calcul taskurile periodice cu perioade și termene fixe, precum și taskurile sporadice cu date de apariție necunoscute dar cu termene fixe. Aceste proprietăți pot fi atinse printr-o abordare pe niveluri, bazată pe planificarea în timp real a taskurilor și pe nucleul de timp real.

Evaluarea unui sistem de operare în timp real se bazează în principal pe capacitățile de timp real, precum:

- promptitudinea răspunsului dat de către sistemul de calcul;
- predictibilitatea timpilor de execuție ai unui apel la nucleu;
- acordarea politicilor de planificare;
- furnizarea de asistența pentru depanarea programului în contextul de timp real atunci când aplicația rulează în câmp (pe teren);
- înregistrarea performanțelor pentru studii de caz.

Se vor dezvolta două aspecte:

Promptitudinea unui răspuns. Promptitudinea unui răspuns al nucleului de timp real poate fi evaluată de către doi parametrii, întârzierea întreruperii și întârzierea de răspuns.

1) Întârzierea întreruperii este întârzierea dintre sosirea unui eveniment în aplicație și momentul în care este acest eveniment este înregistrat în memoria calculatorului. Întârzierea de răspuns este acea întârziere care apare între momentul când sosește un eveniment în aplicație și rapiditatea cu care acest eveniment este procesat de către taskul corespunzător.

2) Predictibilitatea timpilor de execuție a unui apel de nucleu. Un nucleu de timp real include un set de metode pentru reducerea întârzierii, care sunt: reentranța, suspendarea

(preemption), planificarea priorității și moștenirea priorității. Prin urmare, timpul de execuție al fiecărui apel al nucleului poate fi evaluat exact atunci când este executat pentru taskul cu prioritatea cea mai mare. Acest timp este suma dintre apelul propriu-zis și întârzierea dată de secțiunea critică cea mai mare din nucleu.

IV. NUCLEUL DE TIMP REAL

Nucleul de timp real RTX permite o planificare flexibilă a resurselor sistemului precum UCP-ul și memoria oferind câteva moduri de comunicație între task-uri.

Programele scrise pentru nucleul de timp real RTX utilizează constructori în C standard și sunt compilate cu Uneltele de Compilare RealView furnizate de către Kitul de Dezvoltare MDK-ARM.

În plus față de limbajul în C ne mai permite să declarăm foarte ușor funcțiile taskurilor cât și scrierea de programe de timp real ce necesită doar includerea în programul nostru a unui fișier header special și să facem legătura cu librăria RTX.

RTX furnizează funcționalitatea de bază pentru a porni și a opri taskurile (procesele) concurente. Oferă funcții adiționale pentru comunicațiile interprocese. Se pot utiliza funcții de comunicație pentru a sincroniza diferitele taskuri, gestiona resursele comune (ca perifericele și regiunile de memorie) și de a trimite mesaje complete între taskuri.

Se pot utiliza funcțiile de bază pentru a porni executivul de timp real (Real Time Executive), pentru a porni și a opri taskurile, pentru a trece controlul de la un task la altul (planificare round robin). Se pot de asemenea asigna taskurilor priorități de execuție. Când există mai mult de un task în lista READY, nucleul RTX utilizează prioritățile de execuție pentru a fi selectat următorul task de executat (planificare cu suspendare).

Nucleul RTX este însoțit de drivere pentru perifericele de comunicație după cum urmează:

- comunicația CAN;
- comunicația serială;
- comunicația USB;
- comunicația Ethernet și de un sistem de fișiere pentru memoriile flash și de tip SD.

V. ARHITECTURA SOFTWARE A UNUI MASTER INTELIGENT

Arhitectura software propusă este structurată pe taskuri, așa cum se prezintă în Figura 2.

Taskurile propuse sunt grupate pe trei categorii:

- taskuri pentru comunicații;
- taskuri pentru implementarea unor protocoale;
- taskuri pentru administrarea dicționarului de obiecte, comunicația cu PC-ul și lucrul cu fișierele de istorice și configurare.

Task-ul de comunicație CANOpen utilizează două tipuri de obiecte:

- *PDO (Process Data Object)* – obiecte pentru schimbul de date în timp real la nivelul proceselor
- *SDO (Service Data Object)* – obiecte pentru configurare și service la nivelul nodurilor locale.

Pentru obiectele de tip PDO se utilizează modelele de comunicație *producător-consumator* și *master-slave*, în timp ce pentru obiectele de tip SDO se utilizează modelul de comunicație *client-server*.

Taskul de comunicație CAN implementează un driver pentru comunicația CAN, driver care ascunde particularitățile controlerului CAN de pe microcontroler. Recepționează toate mesajele de pe rețea și le trimite taskului care implementează protocolul CANopen Master, dacă sunt mesaje de administrare de rețea sau le trimite taskului care gestionează dicționarul de obiecte dacă sunt mesaje de tipul SDO sau PDO.

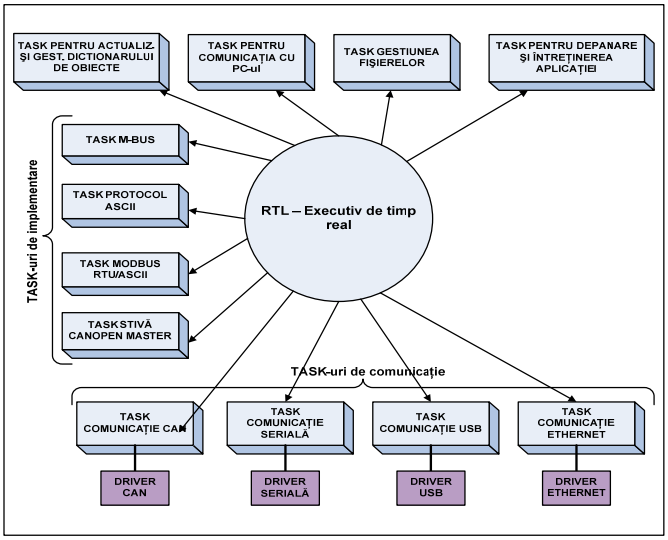

Fig. 2. Schema principală pentru un sistem inteligent pentru achiziția în timp real a datelor distribuite.

Taskul de comunicație serială implementează un driver pentru comunicația serială. Driverul serial folosește 2 buffere circulare, unul de transmisie și unul de recepție. Transmiterea și recepționarea mesajelor se va realiza în funcție de protocolul de comunicație serial activ la momentul respectiv. S-a avut în vedere următoarele protocoale:

- protocolul MODBUS cu modurile de transmisie RTU și ASCII;
- protocolul M-Bus utilizat pentru supravegherea consumurilor energetice;
- protocoale ASCII.

Structura ciclului de achiziție executat de taskurile de comunicație, mai puțin CANopen, este prezentată în Fig.3.

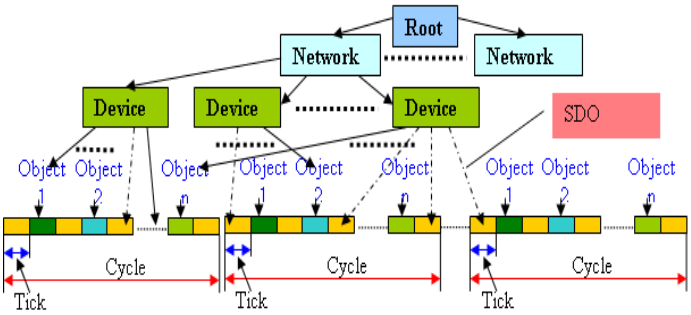

Fig. 3. Structura ciclului de achiziție executat de taskurile de comunicație, mai puțin CANopen.

Taskul de comunicație USB implementează un driver pentru comunicația USB. Acest driver realizează comunicația cu PC-ul prin intermediul dispozitivului USB.

Taskul de comunicație Ethernet implementează un driver pentru comunicația Ethernet. Driverul realizează comunicația cu PC-ul, bazat pe protocolul (stiva) TCP/IP, unde performanțele sunt măsurate din punctul de vedere al fluxului de date, atunci când se transferă blocuri mari de date.

CANopen are definit un management al rețelei (Network Management - NMT) pentru a monitoriza şi controla toate dispozitivele care sunt conectate pe rețea. Fiecare dispozitiv CANopen implementează o maşină de stare, care poate fi controlată prin intermediul magistralei de către Masterul NMT. Această "maşină cu stări" este partea centrală a funcționării Slave-ului NMT. Master-ul NMT este singurul dispozitiv care este capabil să influențeze alte maşini cu stări din rețea şi este permis doar un singur Master NMT pe rețea. De notat că un Master NMT disponibil face parte din resursele rețelei şi poate coexista cu un Slave NMT pe acelaşi dispozitiv fizic.

Spre deosebire de protocolul CANopen, protocoalele MODBUS RTU/ASCII, Protocol ASCII şi M-BUS nu specifică în mod separat comportarea dispozitivului Master. Ca urmare, aceste taskuri au fost concepute doar pentru configurarea si administrarea taskurilor care realizează comunicația propriu-zisă. Sarcinile acestor taskuri sunt următoarele:

- definesc structura ciclului de achiziție evidențiindu-se cuantele pentru PDO-uri şi respectiv SDO-uri;
- definesc mărimea cuantei de achiziție şi numerotarea acestora;
- ataşează fiecărei cuante PDO comanda specifică protocolului şi se indică dacă pentru respectiva comandă se aşteaptă sau nu răspuns;
- defineşte, dacă este cazul, suite de cuante consecutive pentru comenzi multiple.
- Aceste sarcini le realizează în două moduri:
- pe baza unei structuri memorate într-o memorie flash sau sub forma unui fişier stocat fie într-o memorie flash fie pe un suport extern (SDCard, MMCCard, etc.).
- pe baza unor comenzi primite de la calculatorul gazdă, în faza de inițializare a Masterului inteligent. Protocolul nu porneşte în cazul în care nu este disponibilă configurația de lucru (local sau de pe calculatorul gazdă).

Taskurile pot îndeplinii şi funcții de supraveghere cum ar fi:

- contorizarea mesajelor corect transmise respectiv recepționate;
- contorizarea mesajelor cu erori la emisie şi la recepție;
- contorizarea timeout-urilor;
- contorizarea timpului de funcționare, păstrarea evidenței stațiilor care au intrat şi au ieşit din rețea.

Aceste funcții pot genera fişiere de tip LOG stocate local sau pot trimite mesaje de comandă către calculatorul gazdă.

Aceste taskuri, ca şi cel corespunzător protocolului CANopen, nu vor interpreta fişierele de tip EDS deoarece această sarcină necesită resurse sporite şi poate fi uşor de implementat pe calculatorul gazdă.

Din punctul de vedere al Masterului Inteligent, Dicționarul de obiecte constă din mesaje, de lungime variabilă, specifice fiecărui protocol. Pentru păstrarea acestor mesaje se folosesc două buffere circulare de mărime adecvate, unul pentru emisie către calculatorul gazdă şi altul pentru recepție. Operațiile asupra bufferului de recepție sunt realizate de:

- taskul de comunicație cu PC-ul care depune mesajele în dicționar (buffer);
- taskul pentru actualizarea şi gestiunea Dicționarului de Obiecte care preia mesajele de date şi le trimite taskurilor de comunicație serială, CAN sau Ethernet
- si mesajele de control pe care le trimite taskurilor de implementare.

Task-ul pentru comunicația cu PC-ul gestionează două buffere circulare, unul pentru emisie şi unul pentru recepție a căror mărime depinde de tipul de comunicație ales dintre comunicația serială pe RS232, USB sau Ethernet.

Dacă există suport pentru fişier, task-ul pentru gestionarea fişierelor gestionează următoarele tipuri de fişiere:

- fişiere care conțin configurația ciclului de achiziție;
- fişiere care conțin log-uri;
- fişiere care realizează istorice cu mesajele de date din Dicționarul de Obiecte (selectiv pentru fiecare protocol în parte sau fără selecție).

Taskul pentru depanare şi întreținere a aplicației va realiza următoarele funcții de bază:

- gestionează contoarele utilizate de celelalte taskuri realizând fişiere de tip log sau trimițând mesaje de control către taskul de comunicație cu calculatorul gazdă.
- ia decizii privind oprirea sau pornirea unui protocol în funcție de numărul de erori.
- gestionează mesajele de tip spion utile pentru depanarea aplicației.

VI. CONCLUZII

Un obiectiv principal al acestui articol este acela de a prezenta structura arhitecturii unui dispozitiv de tip Master Inteligent care lucrează în timp real care, în funcție de variantă, oferă un subset sau întreg setul de facilități:

- va fi extern calculatorului gazdă;
- va permite conectarea la calculatorul gazdă utilizând interfețele RS232, USB şi Ethernet;
- va implementa următoarele protocoale de rețea: MODBUS TSP/IP Master şi SLAVE, MODBUS RTU şi ASCII MASTER, CANOpen MASTER, M-Bus MASTER, ASCII MASTER;
- va permite cuplarea modem-urilor radio:XTREAM pe 2,4 GHz, GSM, modemul Sony – Ericson;
- va implementa funcția de data logger (va creea şi memora istorice);
- va implementa: stiva TCP/IP şi socket-uri, protocoalele

SLIP şi PPP, protocoalele pentru pagini WEB (HTTP) protocoalele pentru e-mail-uri, protocoalele pentru transfer de fişiere, protocoalele pentru modem-urile radio;

- va implementa componenta de comunicaţie pentru conexiunea cu serverele OPC DA 2.05 sau OPC XML-DA.
- Masterele inteligente pot fi:
- locale, aproape de calculatorul gazdă, sau
- plasate la distanţă (dacă implementează stiva TCP-IP şi alte protocole de la nivelul aplicaţie (HTTP, servere WEB, Modbus TCP-IP, FTP, protocoale de e-mail, etc.)).

REFERINŢE BIBLIOGRAFICE

[1] A. Bhattacharyya (Ed.). Real-Time Systems (Specific Closed Workshop). Esprit PDCS Workshop Report W6, Department of Computer Science, University of York, United Kingdom, Sep. 1990.

[2] A. Burns. Scheduling hard real-time systems: a review. IEE Software Engineering Journal, 6(3):116-128, May 1991.

[3] H. Kopetz. Fault Tolerance in Real-Time Systems. In IFAC World Congress, volume 7, pages 111-118, Tallinn, USSR, Aug. 1990.

[4] H. Kopetz, R. Zainlinger, G. Fohler, H. Kantz, P. Puschner, and W. Schiitz. The Design of Real-Time Systems: From Specification to Implementation and Verification. lEE Software Engineering Journal, 6(3):72-82, May 1991.

[5] H. Kopetz, R. Zainlinger, G. Fohler, H. Kantz, P. Puschner, and W. Schiitz. An Engineering Approach to Hard Real-Time System Design. In Proc. of the Third European Software Engineering Conference, ESEC '91, pages 166-188, Milano, Italy, Oct. 1991.

[6] G. Le Lann. Critical Issues in Distributed Real-Time Computing. In Workshop on Communication Networks and Distributed Operating Systems within the Space Environment, pages 90-105. European Space Research and Technology Center, Oct. 1989.

[7] J.-C. Laprie. Dependability: A Unifying Concept for Reliable Computing and Fault Tolerance. In T. Anderson, Editor, Dependability of Resilient Computers, pages 1-28. BSP Professional Books, Oxford, U.K., 1989.

[8] J.-C. Laprie, Editor. Dependability: Basic Concepts and Terminology, volume 5 of Dependable Computing and Fault Tolerance. Springer-Verlag, 1992.

[9] J. A. Stankovic. Misconceptions about Real-Time Computing: A Serious Problem for Next-Generation Systems. IEEE Computer, 21(10):10-19, Oct. 1988.

[10] J. A. Stankovic and K. Ramamritham. IEEE Tutorial: Hard Real-Time Systems. IEEE Computer Society Press, Washington, D.C., USA, 1988.

[11] A. Wellings. Editorial: real-time software. lEE Software Engineering Journal (Special Issue on Real-Time Software), 6(3):66-67, May 1991.

[12] K. Jeffay. Scheduling sporadic tasks with shared resources in hard-real-time systems. In *Proceedings of the 13th IEEE Real-Time Systems Symposium,* pages 89-99, December 1992.

[13] Houssine Chetto and Maryline Chetto. Some results of the earliest deadline scheduling algorithm. IEEE Transactions on Software Egineering, 15(10):1261-1269, October 1989.

[14] Michael L. Dertouzos and Aloysius K. Mok. Multiprocessor on-line scheduling of hard-real-time tasks. *IEEE Transactions on Software Engineering*, 15(12): 1497-1506, December 1989.

[15] R. Rajkumar, L.Sha, and J.P.Lehoczky. On countering the effects of cycle-stealing in a hard real-time environment. In *Proccedings of IEEE Real-Time Systems Symposium*, pages 2-11, December 1987.

[16] D. B. Stewart and P. K. Khosla. Real-time scheduling of sensor-based control systems. In *Eighth IEEE Workshop on Real-Time Operating Systems and Software*, May 1991.

[17] Karsten Schwan and Hongyi Zhou. Dinamic scheduling of hard real-time tasks and real-time threads. *IEEE Transactions on Software Engineering*, 18(8):736-748, Aug. 1992.

[18] Wei Zhao, Krithi Ramamritham, and J. A. Stankovic. Preemptive scheduling under time and resource constraints. *IEEE Transactions on Computers*, C-36(8):949-960, August 1987.

[19] L. Sha, J. P. Lehoczky, and R. Rajkumar. Solutions for some practical problems in prioritized preemptive scheduling. In *Proceedings of IEEE Real-Time Systems Symposium*, 1986.

[20] GAITAN, Ioan UNGUREAN, „The Uniform Engineering of Distributed Control Systems Using the OPC Specification", Advances in Electrical and Computer Engineering, Volume 8, Number 2, 2008, pag 71-77, ISSN 1582-7445

[21] Vasile Gh. GAITAN, Valentin POPA, Ioan UNGUREAN, Nicoleta-Cristina GAITAN, „The Integration Of Real Device Capabilities In Distributed Applications Based On OPC Tehnology", 12th WSEAS International Conference on COMPUTERS, pp. 48-153, Heraklion, Creta Grecia, 22-25 iulie 2008 ISSN: 1790-5109, ISBN: 978-960-6766-85-8, 2008

[22] Vasile Gh. GAITAN, Ioan UNGUREAN, Nicoleta-Cristina GAITAN, Valentin POPA, KEEPING INDUSTRIAL SYSTEMS AND COMMUNICATION UP-TO-DATE USING INTEROPERABLE COMMUNICATING COMPONENTS AND ELECTRONIC DATA SHEET. pag. 389-396, IEEE Eurocon 2009, may 18-23 2009, Saint Petersburg, Rusia, ISBN 978-1-4244-3861-7

[23] Vasile GAITAN , Valentin POPA, Nicoleta-Cristina GAITAN, Mihai Gabriel DANILA, „MCPI Application – A scalable Human – Computer Interaction (HCI)", Proceedings of ED-MEDIA 2008 World Conference on Educational Multimedia, Hypermedia & Telecomunications, June 30-July 4, 2008 Vienna, Austria, pg. 1522-1527, ISBN: 1-880094-62-2.

[24] Vasile Gheorghita GAITAN, Mihai Gabriel DANILA, Mihai Gavril ROBU, Nicoleta-Cristina GAITAN , „MCPI – a HMI Application for Monitoring and Controlling Industrial Processes", The 9th International Conference on Development and Application Systems, 22-24 May, 2008 SUCEAVA, ROMANIA, ISSN 1844-5020, pg.9.

[25] Ioan UNGUREAN, Alexandru GOLOCA, Nicoleta-Cristina GAITAN, Vasile BALAU - The Implementation of SCADA Applications Based on OPC and Electronic Device Description Technologies, International Symposium on Automatic Control and Computer Science, Iasi, Romania, November, 2007

[26] Nicoleta-Cristina GAITAN, Stefan Gheorghe PENTIUC, Vasile Gheorghita GAITAN, Ioan SARAMET – „The implementation of the Communication Component for Heterogeneous Wide-area Distributed Systems", vol. Sisteme Distribuite, 2008, Suceava, ROMANIA, ISSN/ISBN: 1842-6808, pp.79-84.