Introducere în problematica sistemelor NESECVENȚIALE

Anca Vasilescu, Oana Georgescu

În acest serial ne propunem să parcurgem împreună câteva elemente definitorii ale conceptelor de concurență, paralelism și distribuire, concepte foarte moderne în programare și în teoria sistemelor. Aceste elemente vor fi alese cu grijă, astfel încât să vă trezească interesul și curiozitatea pentru această lume fascinantă.

Pentru început, pentru că sunteți proaspăt întorși din vacanță (și tocmai ați renunțat la modelul distribuit al clasei voastre pentru unul puternic cuplat, în incinta școlii) să amânăm abordarea standard: definiție, definiție, propoziție, teoremă, demonstrație… și să alegem câteva probleme clasice care confirmă că este necesar ca informaticianul să aibă la îndemână pentru modelarea lumii înconjurătoare și alte mecanisme decât cele secvențiale. (Un sistem este secvențial dacă activitatea lui se poate împărți în etape și dacă aceste etape se succed în timp într-o ordine unică, predeterminată algoritmic.)

Jocul vieții

S-ar putea ca această problemă clasică de concurență să vă fie deja cunoscută din [1, pag. 113]. Pentru scopul acestui serial preluăm formularea problemei din [2]:

Un tabel dreptunghiular, notat A, cu m linii și cu n coloane conține celule care pot fi vii sau moarte. Fiind dată o configurație curentă a tabelului, se cere să se determine configurația generației următoare, știind că celulele moarte își păstrează starea, iar o celulă vie moare dacă și numai dacă numărul celulelor vii, vecine ei este mai mic decât 2 sau mai mare decât 4.

O celulă are cel mult 8 vecini: pe orizontală, pe verticală și pe diagonale.

Preferăm să nu comentăm acum această problemă, dar vă invităm să consultați problema 24 din [1, pag. 113] și să încercați să o rezolvați cu mijloacele și structurile capitolului respectiv.

Apoi, după ce ați citit acest serial, vă propunem să o rezolvați ținând cont de elementele specifice pe care le-ați învățat.

Problema rezervării biletelor

Fiecare dintre noi am fost puși în situația de a cumpăra bilete pentru o călătorie pe care o așteptam cu nerăbdare. Pentru aceasta, ascultând sfatul celor mai în vârstă, v-ați preocupat din timp și ați mers cu încredere la o agenție de bilete potrivită tipului de călătorie (cu trenul, cu avionul ș.a.m.d.).

Parcă având biletele în mână, încrederea în reușita excursiei era și mai mare… dar, o dată urcați în mijlocul de transport, surpriză… mai sunt și alți călători care au bilete rezervate pe aceleași locuri ca și voi!

Primul gând se îndreaptă cu reproș spre vânzătoarea de la casa de bilete: oare n-a notat când a rezervat prima dată locurile? Dar dacă mai multe vânzătoare aveau posibilitatea să rezerve aceleași locuri și n-au apucat să se anunțe între ele cine a vândut prima biletele respective? Acum, după plecarea trenului, ar ști să ne spună care locuri au rămas libere și câte bilete s-au vândut?

Această situație, din păcate întâlnită destul de des, este modelul pentru problema rezervării biletelor. O formulare completă ar fi următoarea [2]:

Fiecare terminal al unei rețele de calculatoare este plasat într-un punct de vânzare a biletelor pentru un anumit scop (spectacol, călătorie etc.). Se cere să se găsească o modalitate de a evita vânzarea mai multor bilete pentru același loc.

Acesta este un exemplu de ***paralelism*** deoarece, în același timp (deci în paralel) mai multe case de bilete (terminalele rețelei) vând bilete pentru aceleași locuri.

În plus, apare și ***concurența***, deoarece mai multe ***procese*** (*vânzătoarele de bilete*) încearcă să folosească în același timp un element comun, o ***resursă*** (inițial) comună și anume locul din tren sau din sala de spectacol.

În particular, aici resursa *loc* nu poate fi ***partajată*** (folosită în comun, în același timp, de mai multe procese) deoarece aceasta ar conduce la situația greșită în care mai multe vânzătoare au rezervat același loc pentru călători diferiți.

O astfel de entitate care poate fi folosită exclusiv de un proces la un moment dat este o ***resursă critică***. De exemplu, imprimanta folosită în rețea de mai mulți utilizatori este o resursă critică.

Ce s-ar întâmpla dacă, de exemplu, atunci când imprimanta primește ***simultan*** mai multe fișiere de listat, ea ar lista un singur document în care să ar prelua câteva linii dintr-unul din documentele de intrare, apoi câteva linii din altul, apoi iar câteva din primul, ș.a.m.d?...

Și în acest caz se impune ca, o dată ce imprimanta a început listarea unui document, ea să nu mai preia altă comandă de listare până când nu-și încheie execuția comenzii anterioare.

Un alt exemplu de resursă critică frecvent utilizată de dumneavostră este compilatorul unui mediu de programare folosit în rețea, în același timp, de mai mulți programatori.

Revenind la problema rezervării biletelor, rezolvarea concurenței și implicit a problemei, presupune folosirea unor mecanisme (unelte hard și soft) specializate care să asigure accesul exclusiv al unui proces (*vânzătoare*) la o resursă critică (*loc*) la un moment dat.

Cu aceste mecanisme ne vom întâlni în episoadele următoare.

Problema cititor-scriitor

Problema cititor - scriitor se referă la organizarea unui set de date (o *carte*) când acesta este accesat de procese de tip cititor și scriitor.

Astfel, cartea devine o resursă comună la care au acces diferite procese. Setul de date se numește de obicei *carte* pentru a păstra contextul sugerat de denumirile *cititor* și *scriitor*.

În acest context, un *cititor* este orice proces care poate consulta o resursă simultan cu alte procese *cititor*, dar fără a putea opera modificări asupra resursei respective.

În același timp, un *scriitor* este orice proces care trebuie să aibă acces exclusiv la resursa comună, având și drepturi de modificare asupra resursei respective.

Cu alte cuvinte, este permis ca mai mulți *cititori* să consulte cartea simultan, dar se impune ca atunci când un *scriitor* modifică conținutul cărții, nici un alt proces (indiferent dacă este *cititor* sau *scriitor*) nu trebuie să poată avea acces la carte.

Folosind elementele introduse intuitiv la problema rezervării biletelor, putem spune că în această problemă *cartea* este o resursă critică și că accesul la resursa critică de tip *carte* este permis exclusiv unui *scriitor* sau, în paralel, mai multor *cititori* simultan.

Procesele *cititor* și *scriitor* au de executat activități fundamental diferite, care trebuie ordonate corect pentru execuție. Pentru aceasta, proceselor *cititor* și *scriitor* li se pot asocia anumite reguli de prioritate pentru execuție.

Situațiile de concurență apar când *cartea* este solicitată simultan atât de procese *cititor* cât și de procese *scriitor*. In aceste cazuri programatorul are la dispoziție mai multe opțiuni pentru ordonarea intrării în execuție a proceselor:

- când un *scriitor* solicită accesul la carte, se întrerupe (sau nu) activitatea tuturor *cititorilor* activi astfel încât să poată fi actualizat cu prioritate conținutul *cărții*?
- programatorul întrerupe (sau nu) un *scriitor* care tocmai scrie, în favoarea unui *scriitor* care solicită accesul la *carte*?
- dacă sunt mai multe procese (atât *cititor*, cât și *scriitor*) care așteaptă să primescă accesul la *carte*, care este ordinea în care intră acestea în execuție în momentul eliberării *cărții*?

Dacă veți dori să programați o astfel de problemă, una dintre primele acțiuni va fi să vă hotărâți cum veți trata aceste situații.

Dacă, extinzând modelul la alte exemple, considerăm că resursa comună este un fișier de pe disc, atunci ordinea execuției proceselor trebuie să respecte regula: informația citită trebuie să fie cea mai actuală. Pentru aceasta putem presupune că, de îndată ce un *scriitor* cere să acceseze fișierul (resursa comună), acesta va primi controlul imediat ce toți cititorii curenți (procesele cititor active) și-au finalizat acțiunile, fără ca vreun cititor să mai primească accesul între timp.

Un alt model pe care probabil că îl utilizați frecvent este accesul la un site *Web*. Aici fiecare pagină a site-lui este o resursă critică de tip *carte*, pe care o pot accesa oricâte browser-e simultan (procese client, de tip *cititor*), dar atunci când webmaster-ul (*scriitorul*) modifică conținutul unei pagini, nimeni nu poate citi informația acelei pagini.

Problema producător-consumator

Această problemă modelează situația generală în care avem o succesiune de locații (de exemplu un buffer de memorie sau o bandă rulantă) și două tipuri de procese care au acces la conținutul acelor locații: procese de tip *producător* și respectiv *consumator*.

Un proces *producător* are numai dreptul de a depune un element într-o locație a bufferului, în timp ce un proces *consumator* are numai dreptul de a prelua conținutul unei locații.

Față de problema anterioară în care numai scriitorul avea acces exclusiv la resursa comună, aici și *producătorul* și *consumatorul* trebuie să acționeze individual, la momente de timp diferite, asupra conținutului bufferului. Cu alte

cuvinte, modelarea corectă a problemei trebuie să asigure că atât *producătorul* cât și *consumatorul* au acces exclusiv la resursa comună (buffer) pe durata acțiunii lor specifice (producătorul de a pune obiecte pe bandă și consumatorul de a lua obiecte de pe bandă).

Problema trebuie tratată diferit în funcție de proprietățile particulare pe care le are bufferul în ceea ce privește dimensiunea lui, politica de ocupare cu date, protocolul de acces la date ș.a.m.d.

De exemplu, se poate presupune că bufferul este de dimensiune fixă, adică mărginit sau, mai mult, că "următoarea" locație de accesat este determinată ciclic. De obicei se consideră cazul în care locațiile sunt numerotate și un proces are acces la o anumită locație unic determinată în funcție de operația pe care procesul respectiv o solicită și, eventual, în funcție de operația anterioară care s-a executat asupra bufferului.

De asemenea, putem pune în evidență cazuri particulare de probleme în funcție de numărul proceselor care au acces la bufferul de date. Astfel, putem avea situația mai simplă cu un *producător* și un *consumator* sau, o situație mai generală, cu mai mulți *producători* și mai mulți *consumatori*.

O altă direcție de generalizare rezultă dacă se consideră că un *producător* poate depune pe bandă la un acces mai multe produse și un *consumator*, de asemenea, poate lua de pe bandă la un acces mai multe produse.

În acest model concurența apare deoarece atât *producătorul* cât și *consumatorul* accesează același buffer de date. În plus, apare și necesitatea sincronizării acțiunilor proceselor *producător* și *consumator*.

Aceasta presupune o ordonare în timp a execuției proceselor respective astfel încât să fie îndeplinite condițiile particulare impuse pentru fiecare problemă în parte. În același timp, sincronizarea se impune și pentru a trata cu atenție două situații speciale și anume:

- cazul în care un *producător* încearcă să depună un obiect, dar bufferul este plin, deci nu are unde să îl depună;
- cazul în care un *consumator* încearcă să ia un obiect, dar bufferul este gol.

Importanța acestor situații "limită" reiese din exemplul următor. Să considerăm că la un moment dat un *producător* primește accesul exclusiv la buffer și încearcă să depună obiectul său, dar nu găsește nici o locație liberă în buffer. Mai mult, aplicația este scrisă astfel încât orice acțiune asupra bufferului este suspendată până când *producătorul* transmite un semnal prin care înștiințează sistemul că și-a încheiat activitatea. Dar, bufferul fiind plin, *producătorul* nu-și poate depune obiectul și, implicit, nu ajunge să-și încheie activitatea.

În acest moment apare o configurație nedorită a aplicației și anume situația de interblocare: procesele dețin resursele de așa o manieră încât nici unul nu-și mai poate continua activitatea. Într-adevăr, *producătorul* nu-și poate continua activitatea pentru că bufferul este plin, în timp ce nici un alt proces nu primește acces la buffer până când *producătorul* nu cedează dreptul de folosire a bufferului.

În acest exemplu simplu interblocarea poate fi ușor evitată dacă programatorul dă controlul *producătorului* numai dacă bufferul nu este plin.

Acest model general poate fi "îmbrăcat" în haina interesantă a ***problemei bărbierului somnoros*** [2]:

Prăvălia unui bărbier este formată din două camere: cea de la stradă, folosită ca sală de așteptare și cea din spate, în care se găsește scaunul pe care se așează clienții pentru a fi bărbieriți. Dacă nu are clienți, bărbierul somnoros se culcă. Să se simuleze activitățile care se desfășoară în prăvălia bărbierului.

Aceasta este o reformulare a problemei *producător-consumator* în care locul bufferului de obiecte este luat de scaunul bărbierului, iar consumatorul este bărbierul, în sensul că își bărbierește (consumă) clienții.

Problema grădinii ornamentale

Deși ascunde un model teoretic foarte serios, această problemă poate fi prezentată într-o formă foarte atractivă [2]:

Intrarea în grădinile ornamentale ale unui oraș oriental se face prin N porți. Se pune problema să se țină evidența numărului persoanelor care au intrat în grădină.

Fiecare poartă de intrare în grădină este o resursă care trebuie accesată exclusiv de un proces (de o persoană care dorește să intre în grădină).

Dacă, la un moment, dat pe una dintre porți intră o persoană, atunci în acel moment pe nici o altă poartă nu mai intră vreo persoană în grădină.

Această problemă practică este cea mai apropiată de formularea teoretică, fundamentală, cunoscută ca ***problema excluderii reciproce***, pe care o vom lua în discuție în numerele viitoare.

Problema alocării resurselor

Sunt disponibile **max** *unități dintr-o resursă oarecare. Există* **nc** *clienți care doresc să aibă de mai multe ori acces la resursa respectivă, un acces constând în a lua (a încerca să ia) un număr de unități din resursă, de a folosi aceste unități și apoi de a le returna. Se cere să se simuleze aceste activități.*

Această problemă este una teoretică, foarte importantă, a cărei rezolvare face obiectul unor capitole speciale de matematică aplicată, cum ar fi ***Cercetările Operaționale*** [3].

Modelul problemei este foarte important, el generalizând o problemă de concurență foarte veche și anume ***problema filosofilor chinezi***, cunoscută și sub denumirea de ***problema celor cinci filosofi*** sau problema ***filosofilor la masă***.

Trecând peste ineditul numelui, să vedem la ce se referă această problemă [2]:

La o masă rotundă stau cinci filosofi chinezi. Principala lor activitate este aceea de a gândi, dar, evident, din când în când trebuie să și mănânce, folosind pentru aceasta câte două bețișoare. Stiind că între oricare doi filosofi se află un bețișor și că un filosof poate mânca doar dacă a ridicat de pe masă atât bețișorul din stânga sa, cât și pe cel din dreapta sa, se cere să se simuleze activitățile filosofilor așezați la masă.

Deja din formularea problemei se poate observa că o restricție importantă care se impune este ca la nici un moment de timp să nu avem doi filosofi vecini care să mănânce.

Problema este un caz particular al problemei alocării resurselor în care avem un număr finit de procese (filosofii mâncând) care "împart" un număr finit de resurse (bețișoare), fiecare fiind folosită numai de un proces la un moment dat. O soluție secvențială imediată ar fi următoarea:

```
IaBețișor:
  IaBețișorStâng
  IaBețișorDrept
    Mănâncă
  AșeazăBețișorStâng
  AșeazăBețișorDrept
EndIaBețișor.
```

Pentru această soluție poate foarte ușor să apară ***interblocarea***: dacă toți filosofii doresc să mănânce simultan atunci fiecare ridică bețișorul din stânga sa și așteaptă ca cel din dreapta să elibereze bețișorul pentru ca el să-și poată continua activitatea (adică să aibă în dreapta bețișor pe care să-l ia). Dar cel din dreapta așteaptă același lucru de la cel din dreapta sa și așa mai departe. Astfel, toți filosofii sunt blocați în așteptare și nu se pot continua activitățile.

De aici se vede că această soluție nu este corectă. Ca și alte probleme de concurență, și aceasta trebuie abordată folosind mecanisme specifice intercomunicării între procese.

Problema emițător - receptor

Prezentarea acestei probleme este o ocazie bună pentru noi să aducem în discuție o serie de concepte foarte moderne din sfera rețelelor și aplicațiilor destinate comunicării. În plus, este timpul să spunem câteva cuvinte și despre caracterul distribuit al aplicațiilor nesecvențiale. Modelul general al problemei emițător-receptor este următorul [2]:

Un emițător emite succesiv mesaje, fiecare dintre ele trebuind să fie recepționat de toți receptorii, înainte ca emițătorul să transmită mesajul următor. Se cere să se simuleze această succesiune de activități.

Un caz particular pentru această problemă ar fi cel în care emițătorul este "grăbit" și emite mesajul numai acelor receptori care la momentul emisiei sunt gata să îl primească; totuși, și în acest caz se cere ca cel puțin un receptor să primească mesajul.

Acest caz particular servește pentru a deosebi mai multe tipuri de comunicare (transmitere de date prin mesaje) în funcție de numărul și proprietățile destinatarilor (receptorilor). Astfel, așa cum vom vedea în episoadele următoare, putem avea comunicare *broadcast*, *multicast* sau *unicast*.

În general, problema emițător-receptor este importantă în sistemele în care expeditorul (emițătorul) și destinatarii (receptorii) sunt situați la distanță. În acest caz avem un sistem distribuit în care comunicarea între nodurile din sistem se face prin transmitere de mesaje.

În general, un sistem distribuit este o colecție de procese (emițător, receptor, client, server etc.) împreună cu un subsistem de comunicație între procesele respective. Toate relațiile dintre procese sunt decise pe bază de ***negocieri*** realizate prin schimburi de mesaje.

Am amintit mai sus tipurile de procese server și client. Să spunem câteva cuvinte introductive despre acestea.

Un *proces server* este unul care se oferă să facă servicii altor procese din sistem. Cu alte cuvinte, un server oferă servicii sau produce resurse clienților săi.

În același timp, un *client* este un proces care solicită servicii de la server și le consumă.

Acest model funcționează în toate magazinele: proprietarul magazinului este serverul care oferă produse (resurse), în timp ce cumpărătorul este clientul care consumă (cumpără) produsele oferite de proprietar.

Orice aplicație în care solicitantul acțiunii este un proces și executantul acțiunii respective este un alt proces este o ***aplicație client-server*** [4].

Dacă procesele client și server nu sunt situate pe aceeași mașină (pe același calculator) atunci modelul client-server deservește sistemul distribuit și aplicația devine una distribuită. Este important de subliniat că acest model presupune implicit că, clientul adresează serverului o cerere prin care solicită un anumit serviciu pe care trebuie să îl execute serverul.

Se poate spune că o aplicație client-server include problema emițător-receptor din următorul punct de vedere: atât în momentul în care clientul adresează serverului cererea de a îi oferi un serviciu, cât și atunci când serverul transmite clientului său rezultatul acțiunii solicitate, are loc o comunicare de tip emițător-receptor (sau cazuri particulare ale acesteia).

Din punct de vedere practic, cea mai la îndemână aplicație client-server stă la baza utilizării *Internet*-ului. În 1990 ***Tim Berners-Lee*** a dezvoltat un sistem standard de tip *hipermedia*, numit ***World Wide Web***, pe scurt, ***WWW***. Domeniul de aplicare inițial a fost fizica nucleară, la centrul CERN din Elveția. Esența sistemului consta în faptul că nucleul său îl reprezenta partea de comunicare în rețea și astfel se crease posibilitatea transferului în sistem a paginilor hipermedia, fără restricții.

În 1993 centrul ***NCSA*** (***N**ational **C**enter for **S**upercomputing*), Universitatea din *Illinois*, a dezvoltat browserul *Mosaic*. Importanța și noutatea acestei aplicații era interfața comodă, orientată pe ferestre, care recunoștea transferul paginilor *hipermedia* într-un sistem de informare. Începând cu *Mosaic*, amploarea pe care au luat-o sistemele de tip *WWW* a fost explozivă.

Enorm de multe companii, universități și particulari și-au construit servere de informare care să poată fi citite (vizitate) de oricine posedă un browser.

Fiecare pagină *hipermedia* poate să conțină text, imagini și legături către alte pagini. O astfel de pagină este creată folosind un limbaj de marcare de tip ***HTML*** (***H**yper**T**ext **M**arkup **L**anguage*), care este similar cu limbajul standard *SGML*. În plus față de *SGML*, limbajul *HTML* are pune la dispoziție instrucțiuni referitoare la modul de afișare a paginii create.

Într-un server de informare cum este *WWW* fiecare pagină este identificată printr-un nume, adică o adresă ***URL*** (***U**niversal **R**esource **L**ocator*), care se folosește ca legătură către pagina respectivă.

Dacă o pagină *P-părinte* trebuie să vadă (să conțină o legătură către) o altă pagină, *P-copil*, atunci în pagina *P-părinte* afișată vom avea un text de legătură (*hiperlegătură*) către pagina *P-copil*.

În același timp, codul *HTML* al paginii *P-părinte* va conține atât textul de legătură cât și adresa *URL* a paginii *P-copil*. Deci, în pagina părinte nu apare obligatoriu adresa *URL* a paginii copil. Pentru a deschide pagina copil, este suficient un click de mouse pe textul de legătură din pagina părinte. Pentru a localiza pagina copil, browserul contactează serverul care conține pagina identificată de legătura aleasă pe pagina părinte, transferă informația furnizată de acel server și o afișează în locul paginii părinte.

Protocolul de comunicare respectat de browser și de server este numit ***HTTP*** (***H**yper**T**ext **T**ransfer **P**rotocol*).

Fiecare client (browser) lucrează secvențial, putând să vizualizeze (să afișeze) la un moment dat o singură pagină. Un server oarecare poate să primească concurent oricâte cereri de furnizare de pagini de informații. Deci avem aici un exemplu în care serverul trebuie să deservească simultan, concurent, mai mulți clienți.

O extindere recentă în tehnologia paginilor *Web* este adăugarea de secvențe animate pe pagini, scrise de obicei în limbajul de programare *Java*.

Aici prelucrarea concurentă poate fi realizată la nivelul clientului pentru a mări viteza de animare și pentru a permite animației să co-existe pe pagină în paralel cu acțiunile utilizatorului, de exemplu cu perioada de încărcare a unei forme pe pagina curentă.

Problema bancomatului

Această problemă stă la baza gestionării operațiilor de utilizare a bancomatelor aceleiași bănci de către diferiți utilizatori, simultan și concurent. Enunțul problemei este următorul:

Un client al unei bănci, identificat printr-un număr personal, încearcă să scoată o anumită sumă de bani de la un bancomat. În acest scop, clientul va trebui să introducă mai întâi numărul personal, pe care automatul îl verifică. Dacă numărul personal este corect atunci clientul este solicitat de către automat să introducă suma de bani pe care dorește să o scoată. Dacă, în plus, această sumă solicitată nu depășește valoarea totală a contului atunci clientului i se eliberează de către automat suma dorită. Se cere să se simuleze aceste activități.

Caracterul paralel al problemei este imediat: bancomate diferite pot derula în același timp activități independente. Caracterul concurent al acestei probleme rezultă din faptul că soluția corectă trebuie să rezolve situația în care mai mulți clienți solicită simultan accesul la aceeași bază de date a conturilor aceleiași bănci. Sau, mai mult, clienți diferiți solicită în același timp, de la bancomate diferite, sume de bani din același cont!

Se vede că problema este în strânsă legătură cu teoria bazelor de date, chiar a bazelor de date concurente, în care conceptul de tranzacție este fundamental. Exploatând facilitățile oferite de sistemele de gestiune a bazelor de date pentru asigurarea consistenței bazelor de date la derularea tranzacțiilor, programatorul poate rezolva de o manieră proprie, convenabilă, problema bancomatului, conform cu restricțiile pe care dorește să le impună.

În loc de încheiere, două invitații...

Pentru vizualizarea acestor probleme vă recomandăm să vizitați următoarele pagini *Web*:

- www.csd.uch.gr/~hy345/mirror/workbench/centsync.html
- www.hta-be.bfh.ch/~fischli/kurse/threads
- www.cs.buffalo.edu/~bina/cse421/fall99/lectures/lec5/index.htm.

De asemenea, pentru ca serialul acesta să vă fie cât mai util, nu ezitați să ne contactați la vasilex@info.unitbv.ro. Vă așteptăm cu real interes!

Bibliografie

[1] Rancea Doina, *Limbajul Turbo Pascal*, Editura Libris, Cluj-Napoca, 1994, vol. 1

[2] Georgescu Horia, *Programare concurentă - teorie și aplicații*, Editura Tehnică, București, 1994

[3] Cocan Moise, Vasilescu Anca, *Programarea matematică folosind MS-Excel, Management Scientist, Matlab*, Editura Albastră, Cluj-Napoca, 2000

[4] Boian Florian Mircea, *Programarea distribuită în Internet - metode și aplicații*, Editura Albastră, Cluj-Napoca, 2000

*** *Programare concurentă Cover Story*, PC Report 40 (ianuarie 1996)

*** *Procesarea distribuită Cover Story*, PC Report 50 (noiembrie 1996)

*** *Modelul client/server Cover Story*, PC Report 54 (martie 1997)