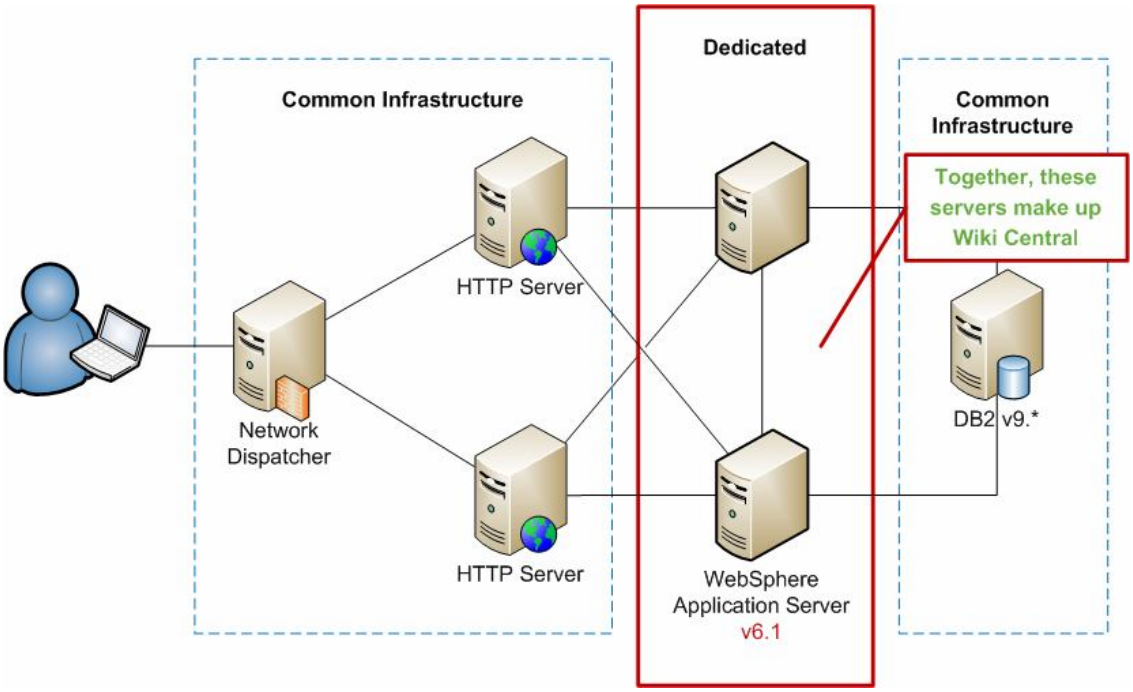
UNIVERSITATEA „POLITEHNICA” din TIMIŞOARA
Facultatea de Electronică şi Telecomunicaţii

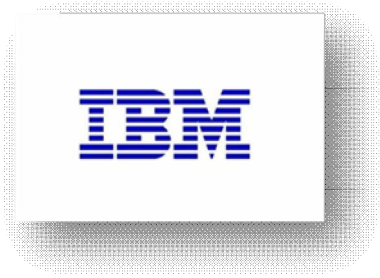
TEZĂ DE DOCTORAT



Doctorand:
Ing. Mircea MIHĂESCU

Conducător ştiinţific:
Prof. Dr. Ing. Horia CÂRSTEA

TIMIŞOARA - 2009



Contribuţii la creşterea disponibilităţii sistemelor digitale complexe utilizând reconfigurarea autonomă dinamică

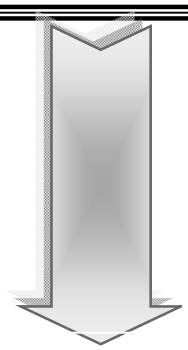
TEZĂ PENTRU OBŢINEREA TITLULUI DE DOCTOR ÎN DOMENIUL INGINERIE ELECTRONICĂ ŞI TELECOMUNICAŢII

Doctorand:
Ing. Mircea MIHĂESCU
IBM, Somers, NY, USA
Director of Web-Ahead,
Development and Security Innovations

Conducător ştiinţific:
Prof. Dr. Ing. Horia CÂRSTEA
Universitatea „Politehnica” din Timişoara
Facultatea de Electronică şi Telecomunicaţii

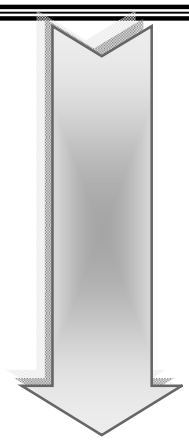
TIMIŞOARA - 2009

CUPRINS



PREFATA

Teza de doctorat intitulată „Contribuții la creşterea disponibilității sistemelor digitale complexe, utilizând reconfigurarea autonomă dinamică”, reprezintă rodul activității de cercetare ştiințifică din ultimul deceniu, desfăşurată de autor, în laboratoarele IBM Toronto, Canada şi New York, USA. În anul 2001, IBM a lansat” Manifestul pentru Calculatoare Autonome „(Autonomic Computing Manifesto), proiect care a generat formarea unor noi colective de cercetare. Activitățile prioritare au fost orientate către soluționarea problemelor specifice echipamentelor electrotehnice autotestabile precum: autoconfigurarea, autorepararea şi autoprotecția. Pe baza preocupărilor anterioare mi-am polarizat atenția şi m-am integrat, în colectivele care cercetau dezvoltarea rețelelor de calculatoare utilizând tehnicile de inginerie software. Aplicarea rezultatelor din teoria controlului statistic au condus la creşterea fiabilității, mentenabilității şi disponibilității sistemelor numerice, prin elaborarea şi dezvoltarea unui concept nou „sistemele autonome de calcul reconfigurabile”. Această lucrare a beneficiat de sprijinul unor persoane cărora doresc să le adresez mulțumiri pe această cale şi să îi asigur de întreaga mea gratitudine: părinților mei: Ion şi Constantina Mihăescu pentru îndemnul continuu de a-mi realiza potențialul maxim profesional, inclusiv prin finaliza acestei lucrări, soției Liliana şi fiicei Irina pentru înțelegere, încurajare si suport, colectivelor departamentelor Tivoli Software Toronto, Web- Ahead New York de la IBM, pentru discuțiile profesionale şi ambianța propice cercetării, creată de-a lungul anilor. Sunt de asemenea recunoscător şi le mulțumesc membrilor Comisiei de recenzare a tezei: Dl. Prof. Dr. Ing. Dan Ionescu de la University of Ottawa, SITE Canada, pentru jalonarea discuțiilor de cercetare şi publicarea rezultatelor parțiale, D^{lui} Prof.Dr.Ing. Paul Svasta de la Universitatea “Politehnica” Bucureşti pentru sprijin şi colaborare, D^{lui} Prof.Dr.Ing. Octavian Prostean de la Universitatea “Politehnica” Timişoara, pentru observațiile şi sfaturile primite pe parcursul elaborării tezei, precum şi tuturor colegilor şi prietenilor care m-au susținut să definitivez teza de doctorat. Nu în ultimul rând, aduc calde mulțumiri profesorului Horia Cârstea, care în calitate de conducător ştiințific, cu pricepere şi profesionalism, a reuşit, pornind de la consultări şi idei, să mă determine să elaborez şi să finalizez această teză de doctorat.



INTRODUCERE

In contextul dezvoltării tehnologice actuale, majoritatea ramurilor producției industriale sunt dependente explicit de sistemele digitale de calcul computerizat. Industria calculatoarelor a evoluat spectaculos în ultimele două decenii, cu avansuri semnificative în domeniile: hardware, sisteme de operare cu aplicații sofware, sau de accesul şi conectarea la internet.

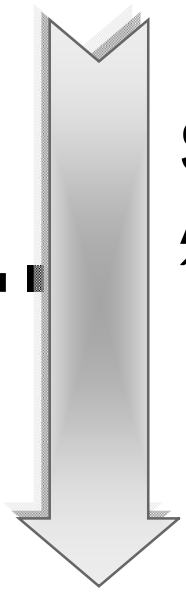
Un mediu computerizat este astăzi complex şi heterogen, incluzând componente hardware şi sofware de la o multitudine de producători şi echipe de surse deschise, ceea ce presupune dificultăți de integrare, instalare, configurare şi menținere în stare de bună funcționare, fără defecțiuni. La rata de creştere actuală a complexității, este posibil ca numai în câțiva ani, mediile computerizate să devină practic imposibil de administrat, chiar de către profesionişti.

Costul ridicat al achiziționării resurselor computerizate a avut ca rezultat generarea unui număr mare de industrii paralele destinate să reducă costul întreținerii şi prin aceasta reducerea impactului general al investițiilor majore în centrele de operare de date. Câteva exemple semnificative includ : IBM's Autonomic Computing , HP's Adaptive Infrastructure sau MICROSOFT's Dynamic Systems Initiative. Scopul declarat al tuturor acestor inițiative este reducerea costului operațional general, mergând până la elaborarea de sisteme auto-dirijate, care operează fără intervenția umană. Conceptul urmărit vizează eliminarea erorilor produse de operatorul uman, care au fost identificate ca o sursă majoră de greşeli şi prin aceasta ,se urmăreşte dezvoltarea sistemelor de operare automată.

Conceptul de automatizare a sistemelor digitale de calcul introdus în ultimii ani ,reprezintă o cale de adaptare la modificările sarcinilor de lucru, eliminarea greşelilor din sistem şi evitarea atacurilor la securitate.Oricum, cele mai multe soluții propuse, au ignorat utilizarea teoriei de control ca o modalitate de asigurare teoretică şi practică solidă, bazată pe dezvoltarea resurselor de automatizare computerizată, precum şi a sistemelor cu auto-administrare.

Termenul autoadministrabil conectat la studiul echipamentelor digitale implică soluții moderne de analiză riguroasă a sistemelor autoscalabile având comportament dinamic şi robustețe la modificările operate în timp real. O arhitectură software administrabilă, se caracterizează prin facilități operaționale în care componentele îşi reconfigurează automat conexiunile într-un mod compatibil cu specificațiile arhitecturale ale sistemelor autotestabile. Obiectivul urmărit constă în minimizarea gradului de administrare explicit, în scopul conservării caracteristicilor arhitecturale la nivel de programare sau de interconectivitate.

Teza de doctorat elaborată, prezintă o sinteză a celor mai importante contribuții aduse de autor la creşterea disponibilității şi mentenabilității sistemelor de calcul digitale complexe, utilizând reconfigurarea autonomă dinamică. Ea reprezintă rodul activității de cercetare ştiințifică desfăşurată de autor, în ultimii zece ani, în cadrul Laboratoarelor IBM Canada şi USA, departamentul WebAhead Development and Security Innovations.



1. STRATEGII DE IMPLEMENTARE A TOLERANTEI LA DEFECTIUNI ÎN SISTEME DIGITALE COMPLEXE

Definiția cea mai uzuală dată sistemelor electronice tolerante la defecțiuni, este aceea potrivit căreia sistemul îşi continuă execuția corectă a funcțiilor de intrare/transfer/ieşire, în prezența unei anumite mulțimi de defecțiuni, ce pot apare în timpul funcționării, fără o intervenție corectivă din exterior. Sistemele tolerante la defectări, conform acestei definiții, se bazează pe ipoteza că defecțiunile de proiectare ale sistemului au fost eliminate înainte de punerea în stare de funcționare.

O interpretare mai generală dată toleranței la defecțiuni, presupune ca aceasta să includă şi abilitatea tolerării defectelor de proiectare, nedetectate înainte de utilizarea sistemului. Tolerarea defectelor este privită în sensul luării măsurilor de precauție în vederea localizării automate a elementelor defecte, dezactivării hardware-lui sau software-lui afectat de erori, comutării rezervelor, sistemul continuându-şi funcționarea pe baza modulelor redundante. Pentru ca echipamentul să fie inclus în sfera sistemelor tolerante la defecțiuni, este necesar ca dezactivarea şi comutarea rezervelor să se realizeze automat.

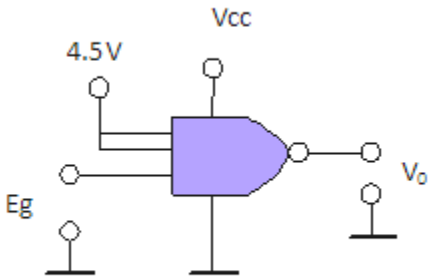
1.1. MODELE DE DEFECȚIUNI PENTRU SISTEME DIGITALE

În practica testării echipamentelor electrice, se urmăreşte elaborarea testelor de verificare pe baza unor modele de defecțiuni, astfel concepute, încât să acopere pe cât posibil o gamă cât mai mare de defecțiuni. Considerând natura potențială a defecțiunilor cu probabilitate mare de apariție, dependente specific de etapele de realizare a unui echipament electronic, de tehnologia utilizată sau de metoda de proiectare adoptată, pot rezulta în general modele de defecțiuni diferite, dintre care vom exemplifica numai două aplicații curente utilizate la testarea echipamentelor electronice numerice.

1.1.1 Modelul de defecțiune "pune pe" (stuck-at)

Este cel mai răspândit model de defecțiune utilizat la testarea circuitelor integrate logice din familia TTL[], conceput pe observația că majoritatea defecțiunilor corespunzătoare acestei tehnologii se manifestă prin blocarea unuia sau mai multor moduri de conexiune la o valoare logică specificată [HC-07].

Modelul de defecțiune *"pune pe"* constă în forțarea intrărilor logice dintr-o familie şi din aceeaşi capsulă la o anumită combinație logică şi verificarea răspunsului logic la ieşire. Fig.1.1a prezintă schema de testare a unei porți logice ŞI-NU din familia TTL, iar Fig.1.1b formele de undă şi amplitudinile: sursei de alimentare Vcc, a generatorului de semnal Eg şi răspunsul corect Vo al porții la ieşire[AF-03].



a) Schema de testare

b) Formele de undă

Fig. 1.1 Aplicarea modelului de defecțiune "pune pe" la testarea unei porti logice

Sursa de alimentare asigură:

$$Vcc = \begin{cases} Vcc_{min} = 4.5V, t < t_1 \\ Vcc_{max} = 5.5V, te(t_1, t_4) \\ Vcc_{min} = 4.5V, t > t_4 \end{cases} \quad (1.1)$$

Generatorul de semnal furnizează:

$$Eg = \begin{cases} Eg_0 = 0V, t < t_0 \\ Eg_1 = 1.4V, t \in (t_0, t_2) \\ Eg_2 = 2.4V, t \in (t_2, t_3) \\ Eg_3 = 1.1V, t \in (t_3, t_5) \\ Eg_4 = 0.7V, t > t_5 \end{cases} \quad (1.2)$$

iar semnalul la iesirea porții trebuie să fie:

$$V_0 = \begin{cases} V_{0H} > 2.4V, t < t_2 \\ V_{0L} < 0.4V, t \in (t_2, t_3) \\ V_{0H} > 2.4V, t > t_3 \end{cases} \quad (1.3)$$

Se constată că acest model aduce poarta în cele mai dificile condiții de funcționare inclusiv prin forțarea intrărilor în zona interzisă unde comportamentul poate fi imprevizibil.

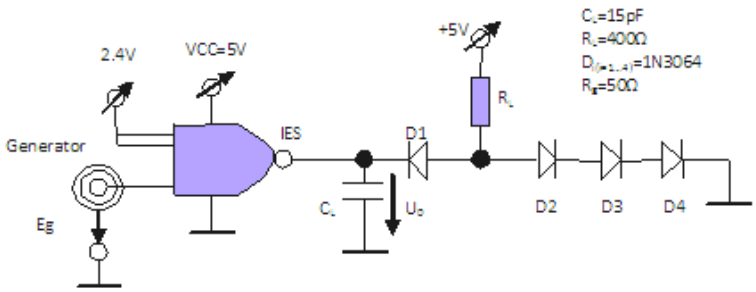
Modelul de defecțiune *"pune pe"* şi-a păstrat utilitatea, chiar cu creşterea complexității circuitelor integrate pe scară largă, pe de o parte datorită simplității (întrucât un test elaborat pune în evidență orice defecțiune potențială), iar pe de altă parte, din cauza eficienței dovedită de experiență şi în detecția altor tipuri de defecțiuni, specifice structurilor numerice moderne.

1.1.2 Modelul de defecțiune de tip *"timp" (timing)*

Testarea parametrilor dinamici ai circuitelor integrate logice şi în general a schemelor numerice, este una dintre cele mai dificile probleme. În principiu, evitarea utilizării circuitelor logice de tip sincron pentru realizarea unui proiect, conduce la minimizarea riscului de apariție a erorilor datorate parametrilor de timp. Totuşi în situația în care comportarea dinamică a schemei este dependentă strict de valorile parametrilor t_{pLH}, t_{pHL} ca şi a întârzierilor produse de liniile de transmisiune, apare necesară elaborarea unui model de tip *"timp"[*BJ-01*].*

Cel mai utilizat circuit care testează defecțiunile de tip timp ce apar în funcționarea porților logice TTL este prezentat în Fig.1.2a şi alăturat în Fig.1.2b sunt prezentate formele de undă ale semnalelor furnizate de generator şi la ieşirea porții testate[CH-05].

Generatorul trebuie să furnizeze semnale cu: E_g *=3,5V;* t_f *=5ns;* t_r *=10ns;* t_w *=0,5ps*, iar la ieşirea porții este necesar ca semnalele să prezinte: V_{OH}>Vo_{Hmin}, V_{0L}<$V0_{Lmax}$, t_{pHL}<*8ns*, t_{pLH}<*12ns.* Creşterea sarcinii capacitive C_L peste valoarea critică C_{L0}*=15pF* conduce implicit la creşterea timpului mediu de propagare pe poartă. Celelalte elemente din schemă: *D1...D4*, C_L R_L simulează încărcarea la ieşire a porții în condiții de FANOUT maxim.



a) Schema utulizată la testare

b) Formele de undă la generator şi la ieşirea porții logice

Fig. 1.2 Aplicarea modelului de defecțiune de tip ”timp” la testarea porților logice TTL

1.2 MODELE DE DEFECȚIUNI PENTRU TESTAREA MEMORIILOR SEMICONDUCTOARE

- **Modelul de bază pentru testarea memoriilor semiconductoare**

Circuitul sub formă de matrice al memoriilor este tratat diferit față de logica obişnuită a cipurilor, atât din punct de vedere al design-ului, cât şi al testării. Structura matriceală a memoriilor este mult mai densă decât arhitectura clasică a chipurilor pentru că memoriile sunt structuri regulate. Acest fapt face ca memoriile să fie mai susceptibile defectelor din siliciu. Regularitatea logicii memoriilor este de asemenea un avantaj, prin faptul că permite memoriilor să fie testate direct, pentru detecția defectelor fizice în loc de detecție de erori, prin utilizarea analizei inductive. Analiza inductivă reprezintă

analiza indiciilor multiple. Această testare este diferită de logica obişnuită de testare, care se bazează pe presupunerea că doar o singură defecțiune este prezentă [DC-05.a].

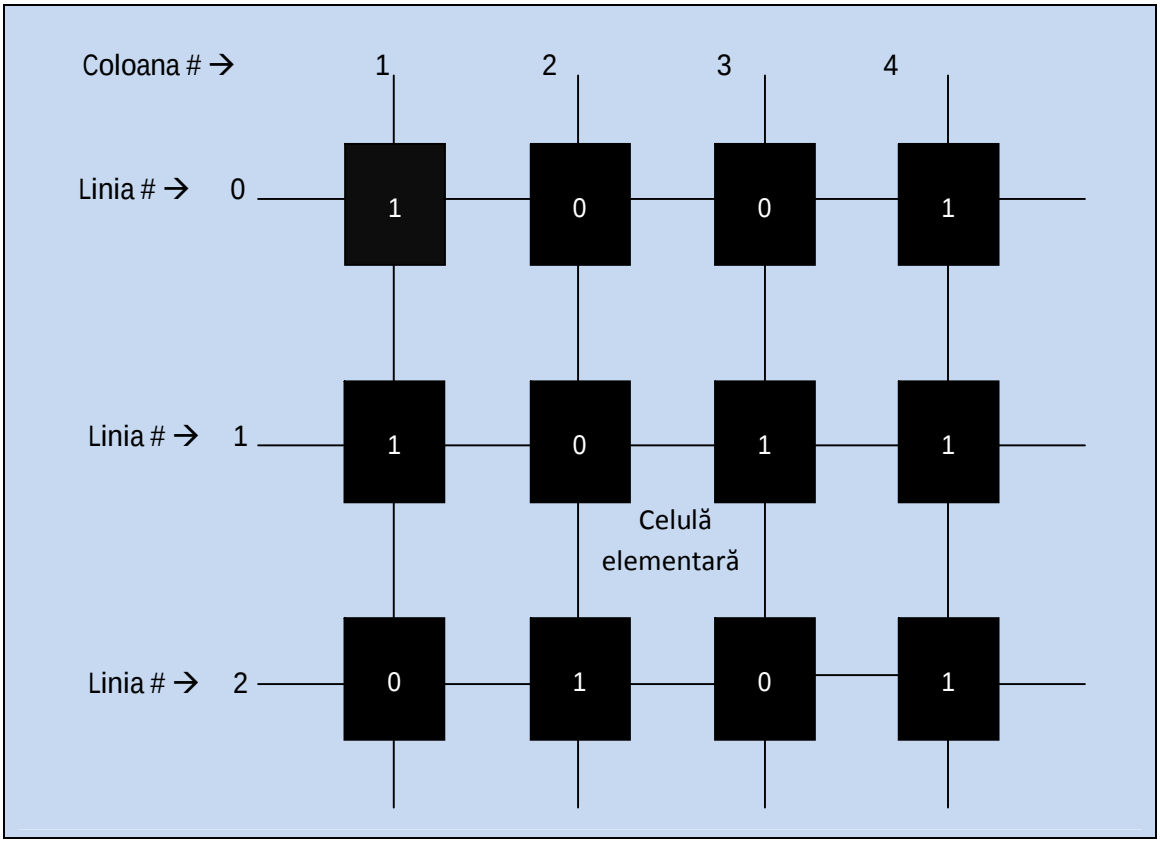


Fig. 1.3 Model simplu al memoriei.

- **Modelul de defecțiune pentru testarea memoriei.**

Modelul de defecțiune de bază consideră că o celulă de memorie, sau bit, nu conține informația corectă în momentul observării (citirii). Pentru detecția defecțiunii, nu contează cîte celule de memorie primesc date eronate; ideea este că memoria conține date eronate. Totuşi, pentru diagnoză, mecanismul defecțiunii este important. Asta înseamna că testarea memoriei este realizată prin plasarea informației într-o celulă de memorie, citirea celulei şi verificarea faptului că informația nu a fost alterată.

Moduri de defectare identificate la testarea memoriei [DG-02.b].

Scopul testării memoriilor semiconductoare constă în aplicarea câtorva date diferite şi secvențe de date memoriei şi stabilirea modului de defectare (cauza defecțiunii) prin identificarea testelor care eşuează şi a testelor care reuşesc.

Modurile de defectare ale memoriilor semiconductoare prin care rezultă date incorecte sunt:

- Stocarea de date: celula de bit acceptă, reține şi returnează informatie;
- Integritatea datelor: informația dintr-o celula elementară nu se degradează in timp;
- Furnizarea datelor: liniile de date pot aplica operații de citire/scriere;

- Decodarea datelor: informația potrivită este furnizată liniilor de date corecte;
- Revenirea pentru date („Data Recovery”): verificarea timpului de acces şi a timpului de revenire a liniilor de date („sense-line recovery time”);
- Furnizarea adreselor: driverele de linie pot aplica selecția de adresă;
- Decodarea adreselor: fiecare cuvânt poate fi selectat, şi asta in mod unic;
- Şuntare („Bridging”): o parte a memoriei nu afectează o altă parte a memoriei;
- Legare („Linking”): o parte a memoriei nu blochează o altă parte a memoriei.

1.2.1 Modelul de defecțiune „pune pe” pentru un bit de memorie

La fel ca şi logica obişnuită de testare, o celulă elementară de memorie poate fi modelată printr-o defecțiune de tip „pune pe”. Cum ariile de memorie sunt structuri regulate, modelul „pune pe” poate fi aplicat oricărei porțiuni din memorie, cum ar fi: unui bit, unui octet, unui cuvânt sau unui bloc de memorie [DH-06].

Aplicarea modelului „pune pe” are la bază inscrierea informației unei celule elementare sau a unui grup de celule la o valoare logică constantă. Pentru celule individuale, valoarea poate fi 1 logic sau 0 logic. Pentru o porțiune mai extinsă de meorie, valorile pot fi toate 1 logic, toate 0 logic, sau o anumită succesiune de valori (de exemplu: 01100011 pentru un octet).

Sursa unei defecțiuni „pune pe” dintr-o celulă elementară poate consta dintr-un defect la unul din tranzistoarele de stocare, de alimentare, de la masă sau de selecție. Defecțiunea „pune pe” este detectată prin scrierea pe rând a valorilor logice 1 şi 0 în fiecare celulă elementară şi apoi citirea valorilor reale din celule şi compararea cu valorile înscrise.

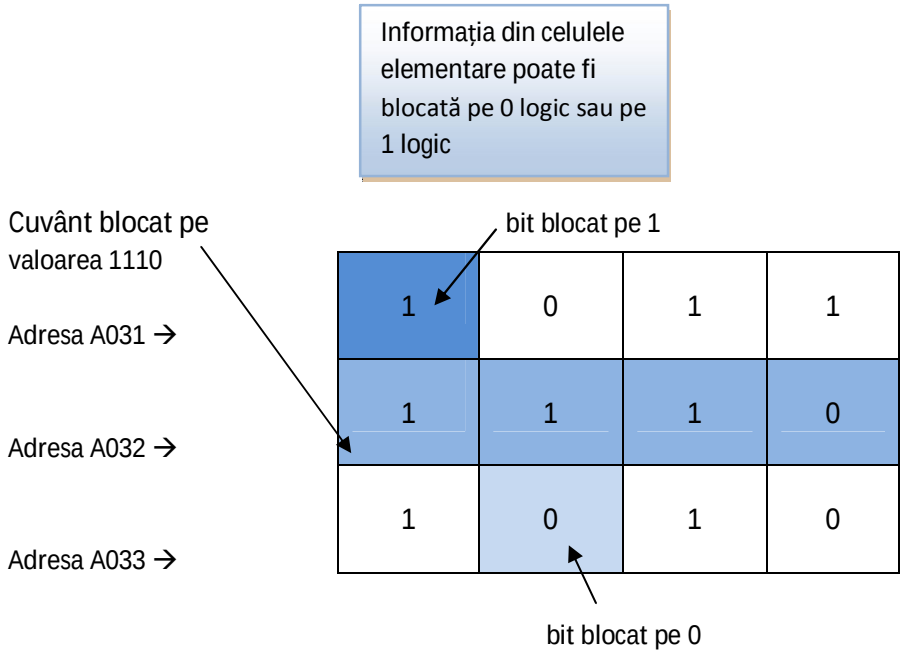


Fig. 1.4 Defecțiuni de tip „pune pe” pentru celule elementare sau grupări de celule.

1.2.2 Modelul de defecțiune de tip *"punte"*

Una dintre cele mai frecvente probleme ce pot apare la ariile de memorie, o reprezintă defecțiunile de tip punte. Densitatea ridicată şi apropierea dintre celulele elementare de memorare conduce la o probabilitate ridicată de apariție a punților fizice dintre celule adiacente. În arhitectura memoriilor celulele adiacente nu sunt totdeauna în ordine din punct de vedere al logicii memoriei. Acest fapt face ca punțile să fie mai greu de depistat, din moment ce doua celule adiacente fizic s-ar putea să nu aibă nici o legătură logică [HC-00] (de exemplu, bitul 4 din cuvântul 3 poate fi plasat fizic lângă bitul 8 din cuvântul 156 dacă memoria este formată din blocuri 4x4).

Defecțiunile de tip „punte" pot fi de tipul 0 Ohm, rezistive, sau de tip diodă. Puntea de tip 0 Ohm poate cauza ca doua celule elementare să reacționeze una în funcție de cealaltă (o schimbare în una produce o schimbare în cealaltă). Puntea rezistivă poate cauza o defecțiune de întârziere (celula are o schimbare întârziată dintr-o stare logică în alta). Puntea de tip diodă poate apărea ca o defecțiune intermitentă, din moment ce doar un element işi va schimba valoarea şi numai când elementul de memorie de la capătul cu sursa al punții, se schimbă (şi numai când informația care se modifică la sursă este diferită de informația prezentă la destinație). O punte de tip diodă bidirecțională este similară cu o punte de tip 0 Ohm. Această multitudine de tipuri de punți, fac ca testul pentru identificarea defecțiunilor de tip *"punte"* să fie foarte complex[JK-05].

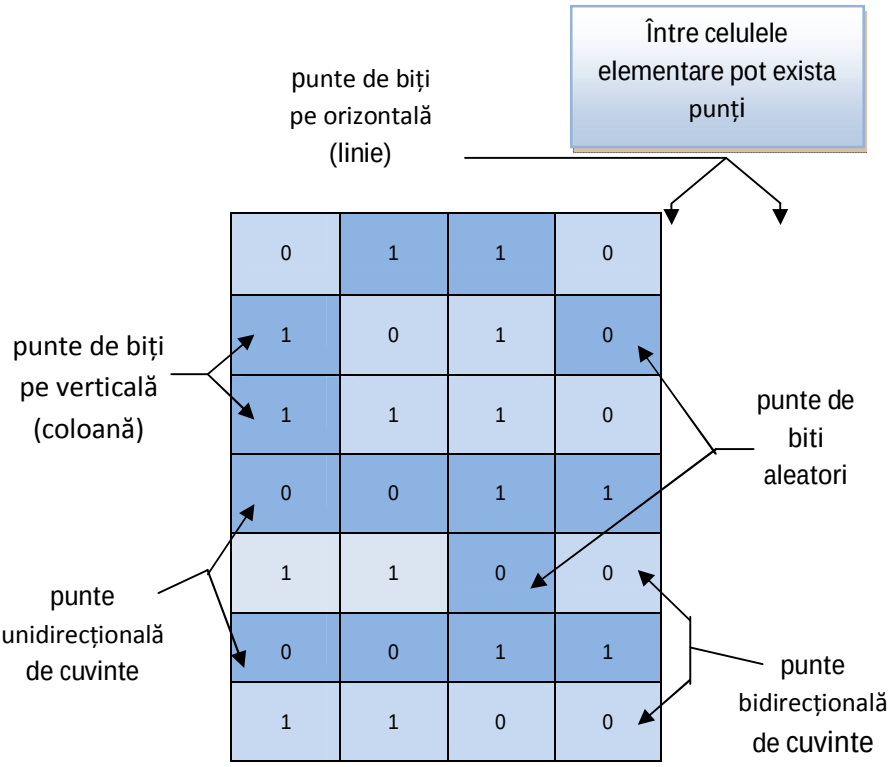


Fig. 1.5 Diverse forme de defecțiuni de tip punte

Defecțiunea de tip legare („linking”) este foarte asemănătoare cu defecțiunea de tip punte, cu excepția faptului că o schimbare a informației dintr-o celulă elementară sau locație de adresă, face ca informația din altă celula elementară sau locație de adresă să „înghete” (rămâne blocată astfel încât informația nu poate fi modificată la un moment de timp ulterior). În realitate, efectul acestei defecțiuni pare a fi cel al unei defecțiuni de tip „pune pe” intermitente – locația de memorie poate trece de test chiar si de câteva ori, până in momentul în care locația conexă este accesată şi date diferite sunt stocate în elementele de memorie adiacente(test eşuat).

- **Detecția defecțiunilor de tip „punte”**

Defecțiunea de tip „punte” este detectată prin scrierea unor prototipuri alternative sau complementare, care vor plasa informații opuse în celule adiacente. Dacă dispunerea fizică a celulelor nu este cunoscută de proiectantul testului, atunci ar trebui aplicate modele generale de fond complementar [HC-07] (0 – F, 3 – C, 5 – A, etc.).

De asemenea trebuie remarcat că punțile unidirecționale de tip diodă, pot fi sensibile la ordinea adreselor. Dacă informația este scrisă şi citită de la adresa 0 la adresa N (N fiind valoarea maximă a adresei, de exemplu 255 pentru o memorie de 8 – biți pe 156 cuvinte), atunci o punte unidirecțională ar putea deteriora informația de la adresele inferioare când se scrie la adresele superioare. De exemplu, scrierea informației în cuvânt la adresa 128 ar putea antrena informația din cuvânt la adresa 3. Dacă testul aplicat este doar de la adresa 0 la 255, atunci aceasta eroare nu ar fi descoperită în timpul testului. Din acest motiv, modelele pentru punți trebuie aplicate în ambele sensuri de parcurgere a adreselor – şi de mai multe ori, deoarece prima accesare, ar putea excita defecțiunea, dar numai accesarea multiplă ar conduce la detecția defecțiunii.

1.2.3 Modelul defecțiunii de decodare

Defecțiunea de decodare a memoriei poate apare la orice nivel (Fig. 1.5). Acest model de defecțiune poate fi aplicat adreselor locale de memorie şi magistralelor de date, adreselor la nivel de sistem şi magistralelor de date, sau la decodarea intra-memorie, care cuprinde driverele linie şi coloană ale matricei interne şi controlul logic citeste-scrie. Modelele de defecțiuni care pot fi aplicate sunt cele de tip „pune pe”, „blocat deschis”, de tip „punte” şi „întârziere”.

Modelele de defecțiuni aplicate semnalelor magistralei de date, driverului decodorului coloană şi driverelor coloană, reprezintă biți dintr-un cuvânt. Aceste modele de defecțiuni se manifestă prin biți blocați pe o valoare logică specificată de-a lungul întregului sau pe anumite porțiuni ale spațiului adreselor (de exemplu, o singură coloană poate controla valorile informației din bitul 27 a întregului spațiu de adrese a uniei memorii extinse pe 32 de biți), sau doi biți de date pot fi legați împreuna, respectiv timpul de acces pentru o linie de date poate fie mărit [JK-05].

Modelele de defecțiuni aplicate magistralelor de adresă, decodoarelor de adresă, decodoarelor de linii şi driverelor decodoarelor de linii, reprezintă spațiul de adrese a cuvintelor într-o memorie. Aceste modele de defecțiuni rezultă prin selectarea aceluiaşi cuvânt sau aceleiaşi adrese, indiferent de adresa aplicată (de exemplu, o singură linie poate reprezenta cuvantul 96 din cele 128 de cuvinte dintr-o

matrice de memorie), sau două cuvinte pot fi selectate simultan, respectiv timpul de acces pentru unul sau mai multe cuvinte poate fi mărit.

- **Detecția defecțiunilor de decodare**

Tipurile de modele de defecțiuni care pot apare, sunt cele de tipul „blocat pe", de tip *"punte"* şi defecțiuni de „întârziere" datorate scurtcircuitelor, întreruperilor, punților şi altor erori de proces. Modelul necesar pentru a permite identificarea şi detecția defecțiunilor de decodare de tipul „pune pe" sunt acele modele care vor aplica 0 şi 1 logic fiecărei informații şi linii de adresă, în aşa fel încât, fiecare adresă poate fi accesată în mod unic (dacă întreaga memorie ar avea înregsitrată doar valoarea de date A, atunci selecția adresei eronate ar trece neobservată). Defecțiunile de „întârziere" şi cele de tip *"punte"* se pot manifesta şi detecta prin aplicarea unor vectori ce produc tranziții logice (perechi de vectori) spațiul de adrese şi liniilor de date. Punțile pot fi testate similar pentru toate celulele elementare, dar modelul de eroare trebuie aplicat doar adreselor liniilor de date şi logicii de decodare [JK-01] (dispunerea fizică a memoriei nu e necesară, deoarece decodarea reprezintă organizarea logică a memoriei).

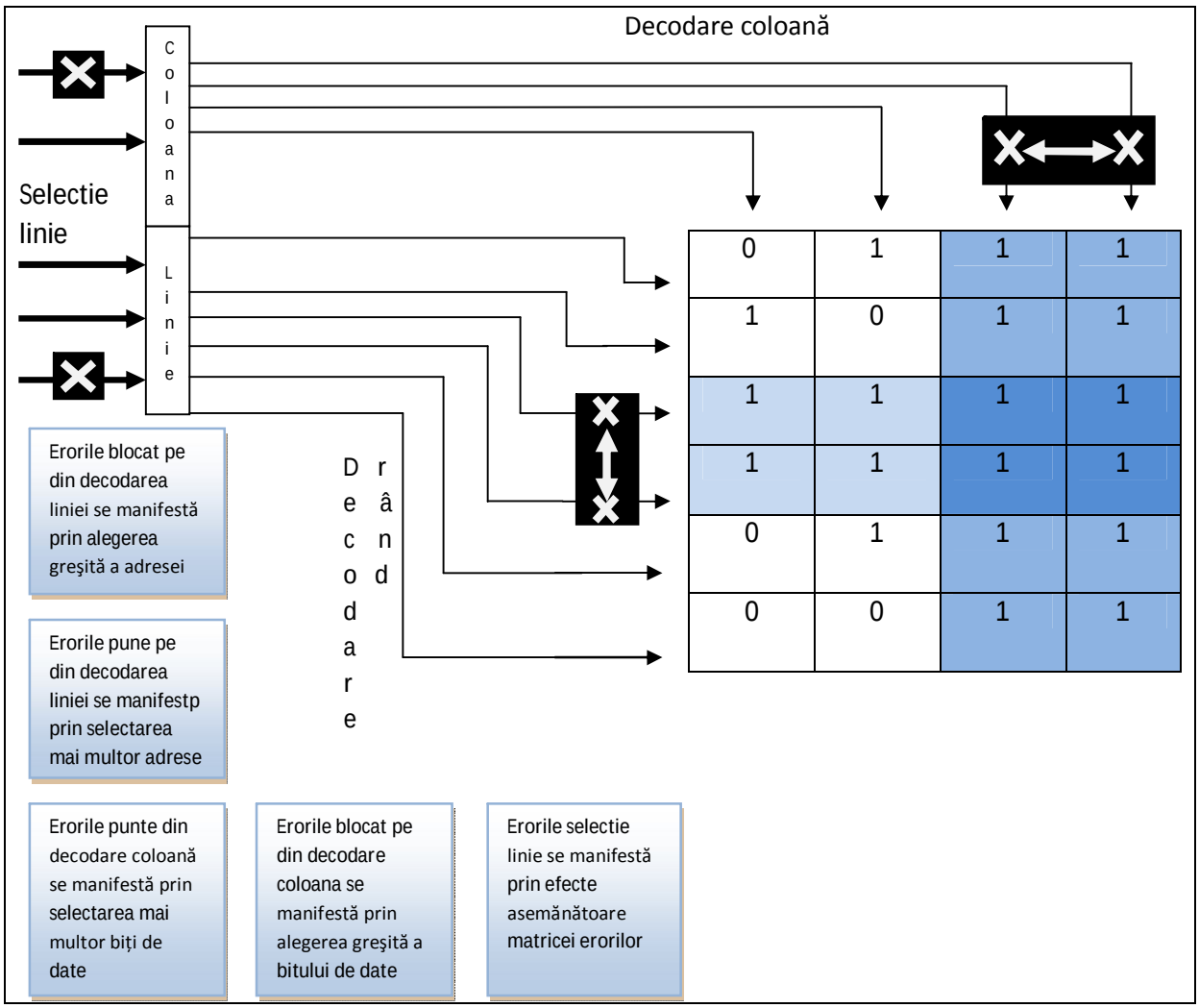


Fig. 1.6 Defecțiuni de decodare

1.2.4 Modelul de defecțiune al integrității datelor.

Una din problemele specifice ariilor de memorie se referă la scurgerile de curent, sau punțile care pot cauza degradarea în timp a informației. Metoda uzuală pentru verificarea integrității datelor (data retention) este de a înscrie date în memorie, a le lăsa acolo pentru o perioadă de timp şi apoi a le citi şi verifica. Pentru a exacerba condițiile testului, o stare forțată de scurgeri poate fi emulată prin deplasarea unui tipar de tip „înconjurat de complement" sau „tablă de şah" în aria de memorie, sau prin modelarea nivelelor tensiune / temperatură în timpul procesului de testare. Odata ce informația a fost plasată în memorie, semnalul de tact poate fi oprit, realizându-se o pauză statică. O altă metodă poate fi plasarea datelor în memorie, urmată de „blocarea la scriere" a memoriei, testarea altor aspecte privind memoria, apoi reîntoarcerea pentru a citi şi evalua memoria la un moment de timp ulterior.

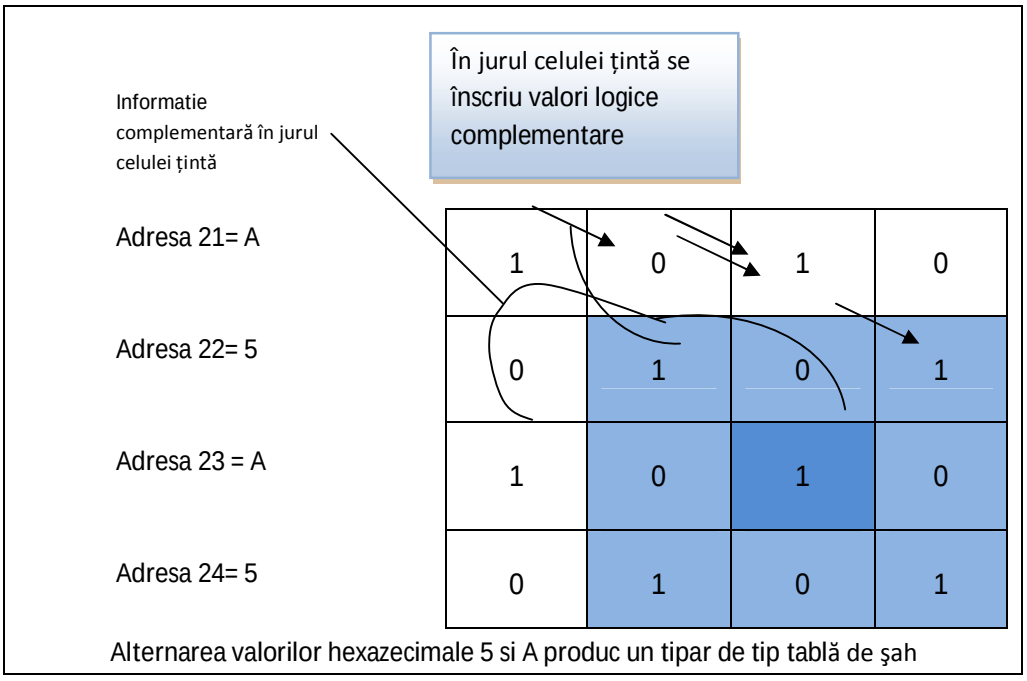


Fig. 1.7 Defecțiuni de integritate a datelor

Memoriile DRAM folosesc condensatorul ca element de stocare. Cum un condensator are oă de timp cunoscută asociată păstrării integrității datelor, este necesară reîmprospătarea ca o operație normală în cadrul funcționării. Aceasta creşte complexitatea testelor pentru DRAM prin faptul ca secvența de testare trebuie de asemenea să permită ciclul de reîmprospătare, iar ciclul de reîmprospătare este acum o operație care trebuie verificată ca făcând parte din procesul de testare. Reîmprospătarea, de exemplu, împiedică citirea şi scrierea continuă de la începutul memoriei şi până la sfârşitul ei, dacă acest proces depăşeşte perioada de reîmprospătare.

1.3 TOLERANȚA LA DEFECȚIUNI ŞI DIAGNOZABILITATE

Pentru definirea toleranței la defecțiuni şi a diagnozabilității vom face următoarele notații:

- (S_p, R, ρ) - defineşte formal o schemă de reprezentare a unui echipament electronic funcțional, unde: S_p – reprezintă clasa specificațiilor sistemului *R* - clasa realizărilor sistemului, iar $\rho: R \rightarrow S_p$ reprezintă funcția de realizare a specificațiilor.
- $(S, F, \emptyset)$ - este schema de reprezentare a unui echipament electronic cu defecțiuni, unde *F* reprezintă mulțimea defectelor posibile.

Considerăm acum un sistem cu defecte $(S, F, \emptyset)$ într-o schema de specificare (S_p, R, ρ) şi un defect impropriu $f \in F$.

Se defineşte o relație de toleranță la defecțiuni, pentru un echipament electronic având o schemă de reprezentare de tipul (S_p, R, ρ) ca fiind o relație structurală *t* între *R* şi S_p , $t \subset R \times S_p$ astfel încât pentru orice $r \in R$, mulțimea $(r, \rho(r)) \in t$.

Cu acest formalism se poate da în continuare următoarea:

Definiție: Dacă (S_p, R, ρ) reprezintă un sistem cu defecte într-o reprezentare $(S, F, \emptyset)$ atunci *t* constituie o relație de toleranță pentru defectul $f \in F$ dacă $(\emptyset(F), S) \in t$, iar atunci când mulțimea $(\emptyset(F), S) \notin t$, defectul este diagnosticat cu relația structurală *t*.

Astfel, se poate constata că un defect impropriu este întotdeauna acceptat de sistem, întrucât $\rho(\emptyset(F)) = S$ de unde rezultă $\rho(\emptyset(F), S) \in t$.

Pe de altă parte, dacă un defect $f \in F$ este diagnosticabil cu relația structurală *t*, înseamnă că va exista o secvență a semnalelor de intrare, care va produce un semnal de ieşire eronat, prin aceasta rezultând în fond un test de diagnosticare a defectelului.

Cele două concepte enunțate: toleranță la defectări şi diagnosticare a defectelor sunt deci complementare. Este posibil însă ca pentru unele sisteme de timp $(S, F, \emptyset)$ să existe o relație *t* de toleranță a defectărilor şi o relație *t'* de diagnosticare a defectelor, astfel încât să existe implicația $t \neq t'$.

1.3.1 Algoritmi de detecție şi diagnosticare a defectărilor

Acțiunile de localizare a defectelor sistemului pot fi considerate ca fiind incluse în algoritmul de diagnosticare a defectărilor, în care se disting următoarele abordări[HC-07]:

- *Testarea inițială* se desfăşoară înaintea punerii în funcțiune a echipamentului şi permite identificarea elementelor hard defecte, introduse în timpul proceselor de fabricație, a erorilor de proiectare sau a erorilor soft.
- *Testarea on line* are loc simultan cu operarea normală a sistemului şi poate fi implementată atât cu mijloace hard cât şi soft. Această operație implică utilizarea codurilor detectoare de erori, dublarea elementelor, compararea variabilelor de ieşire,

utilizarea sistemelor de supraveghere cum ar fi microprocesoarele de mentenanță care execută programe de monitorizare a sistemului.
Principalul avantaj al testării *on line* constă în detecția şi diagnosticarea defectelor înainte de producerea unor prejudicii importante în sistem.

- *Testarea modulelor redundante* trebuie să constate dacă modulele de rezervă sunt capabile să preia funcțiile modulelor funcționale. În acest scop se utilizează fie metodele testării *on-line* (la sistemele autotestabile) fie metodele testării *off-line* (programe de diagnoză pentru mentenanță preventivă) funcție de tipul redundanței protective utilizate.

Fig.1.8 Prezintă ordinograma simplificată a unui algoritm de detecție şi diagnosticare a defectelor.

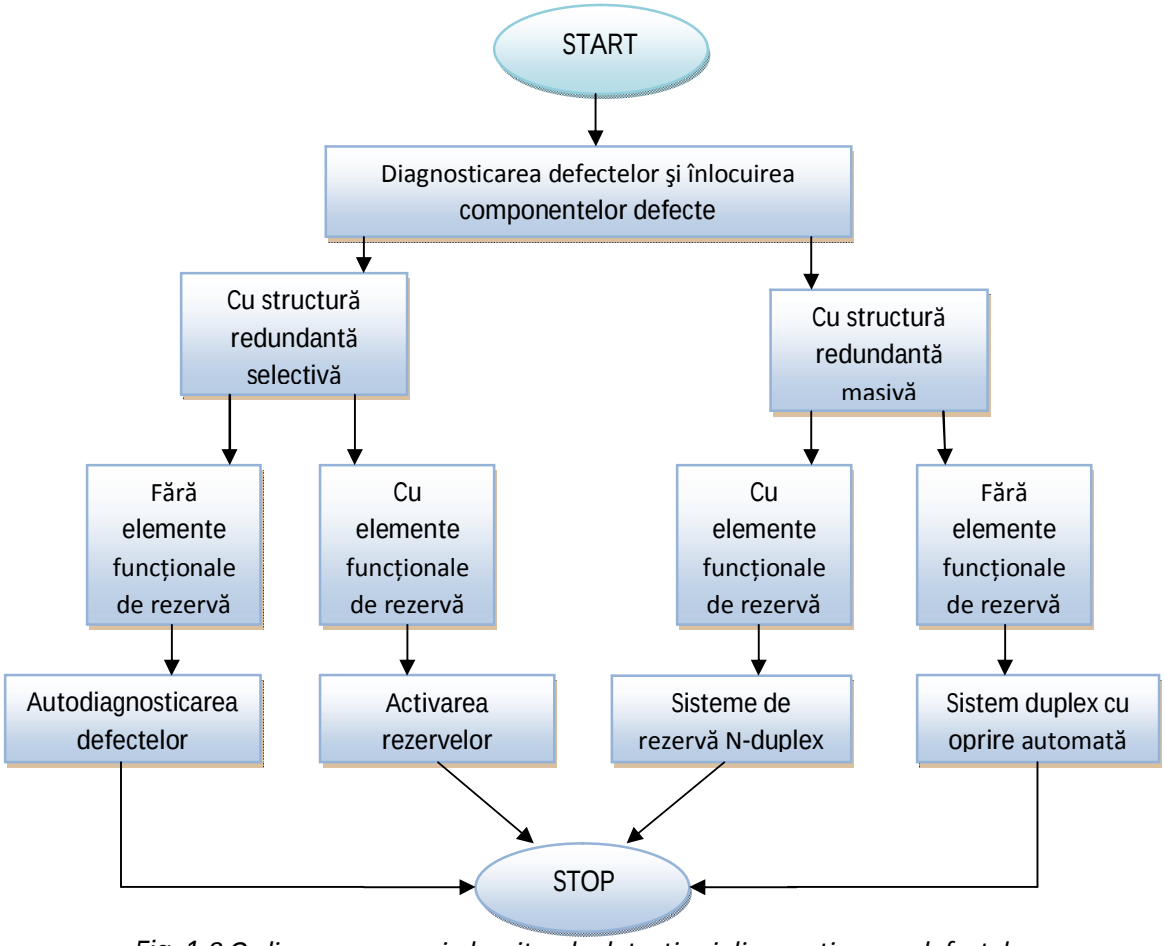


Fig. 1.8 Ordinograma unui algoritm de detecție şi diagnosticare a defectelor

1.3.2 Algoritmi de reconfigurare a sistemelor

Toate acțiunile inițiate din momentul detecției unei defecțiuni şi până la reluarea operațiilor de

funcționare normală a sistemului, constituie etapele algoritmului de reconfigurare a sistemului.

Principalele metode de reconfigurare, funcție de starea sistemului şi în modul interacțiunii cu operatorul uman, pot fi clasifícate ca în Fig. 1.9.

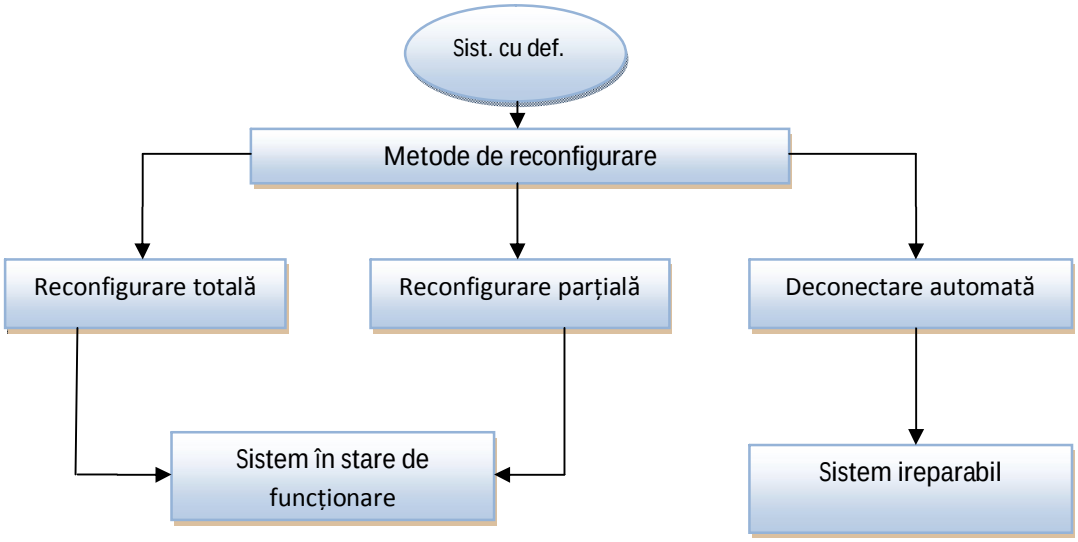


Fig. 1.9 Clasificarea principalelor metode de reconfigurare a echipamentelor electronice

- Reconfigurarea totală a sistemului presupune refacerea structurii hard şi soft existentă înaintea apariției defectului prin înlocuirea elementelor defecte cu rezervele sistemului.
- Reconfigurarca parțială *(fail soft operation)* aduce sistemul la o stare de funcționare corectă dar cu o capacitate de operare redusă.
- Deconectarea automată a sistemului pentru evitarea diferitelor erori şi pentru încetarea interacțiunii cu alte sisteme sau operatori umani, reprezintă un caz limită al reconfigurării parțiale. Totodată în cadrul mentenanței protective se furnizează mesaje de deconectare şi diagnostic operatorilor, sau diferitelor echipamente periferice.

Implementarea reconfigurării la nivel hard implică utilizarea unor circuite specializate în inițializarea procedeelor de reconfigurare [JB-04a].

Reconfigurarca sistemelor la nivel soft, implică utilizarea unor programe adiționale care asigură operarea softului operațional în prezența detectării nivelului corespunzător din hardul defect.

1.3.3 Algoritmi de mascare a defectelor

O formă specială de reconfigurare a sistemelor pentru tolerarea defectărilor, o constituie mascarea acestora, realizată datorită structurii redundante protective de tip static a echipamentului respectiv.

În acest caz, simptomele defectelor prezente în modelele de bază ale sistemului, nu sunt transparente la ieşirile acestuia, atâta timp cât modulele de rezervă sunt în stare de funcționare.

Analizând un sistem care maschează defectele, nu se poate distinge un proces de detecție a defectărilor urmat de procesul de reconfigurare corespunzător. Elementele de rezervă sunt conectate permanent şi ca urmare mascarea defectelor se realizează instantaneu şi automat.

Utilizarea tehnicilor de mascare a defectărilor în implementarea toleranței la defectări, se bazează pe presupunerea că defectările modulelor de rezervă, reprezintă evenimente independente. Aceasta poate constitui explicația faptului că tehnicile de mascare a defecțiunilor nu se justifică să se aplice structurii interne a unui circuit integrat la care este foarte probabilă apariția defectărilor dependente statistic.

1.4 STRUCTURI REDUNDANTE PROTECTIVE

Redundanța este definită ca reprezentând utilizarea într-un sistem, a mai multor elemente decât sunt necesare pentru îndeplinirea funcțiilor acestuia. Scopul declarat îl constituie realizarea unei funcționări corecte chiar în prezența existentei unor defecțiuni.

Elementele suplimentare sunt fie circuite electronice, când se obține o redundanță structurală de tip hardware, fie programe adiționale pentru a obține o redundanță structurală de tip software [JK-04], sau un timp suplimentar de operare pentru a obține redundanța de timp (temporală)[JK-05].

1.4.1 Definirea redundanței protective în sisteme digitale

Redundanța este o metodă comodă de creştere a fiabilității sistemelor electronice, dacă defectările elemenetelor care compun sistemul sunt evenimente independente. Practica însă a demonstrat că dimensionarea sistemelor redundante, în ipoteza simplificatoare a independenței defectărilor, conduce la evaluări inexacte. De aceea pentru o proiectare realistă a sistemelor electronice complexe cu structură redundantă, este necesar să se țină seama de efectul defectărilor dependente de nodul comun la care se raportează analiza. Tehnicile de realizare a redundanței structurale pot fi clasificate astfel:

- *Redundanta de tip static (Tehnici de mascare a defectelor)* implică codarea funcțiilor logice ale sistemului cu coduri redundante, precum şi tehnici de recunoaştere a erorilor şi mascarea instantanee a efectului unui defect din sistem, prin activarea automată a elementelor de rezervă. Această multiplicare a rezervelor în structura sistemului poate fi realizată de la nivelul componentelor până la nivelul întregului sistem (structuri individuale respectiv globale).

Principalele avantaje oferite de aplicarea redundanței de tip static pot fi sistematizate astfel:

- acțiunea corectivă este instantanee;
- nu este necesară o diagnosticare prealabilă a defectelor;
- trecerea de la sisteme nerendundante la acelaşi sistem cu structura redundantă este relativ simplă.

Trebuie remarcat însă că pentru sistemele digitale, structura redundantă de tip static ridică numeroase probleme de sincronizare între modulele sistemului şi introduce numeroase limitări ale *fan-in* şi *fan-out.*

- *Redundanta de tip dinamic (de comutație)* este termenul consacrat tehnicilor care presupun utilizarea mai multor module de rezervă cu aceleaşi funcții, din care doar o parte sunt operaționale, celelalte fiind în stare de veghe (vor fi conectate numai în momentul identificării unor defecțiuni). Această tehnică este utilizată la realizarea autoreparării sistemelor prin comutarea rezervelor în mod automat, sau la realizarea reconfigurării, atunci când sistemul trebuie reorganizat într-o configurație diferită de cea inițială [FK-00].

Utilizarea structurii redundante de tip dinamic prezintă următoarele avantaje:

- necesită să fie alimentată electric numai un modulul de rezervă;
- comutatorul care realizează activarea rezervelor în sistem asigură o izolare a defectelor;
- numărul de rezerve al sistemului poate fi optimizat pentru o misiune dată, fără a face modificări în structura de bază;
- nu introduce limitări constructive cum este cazul structurii redundante de tip static, referitor la echiparea sistemelor numerice.

1.4.2 Structuri redundante protective statice rezultate prin procedee de multiplicare

Aceste modele de structuri redundante se pot aplica fie la nivelul componentelor sau modulelor funcționale, când se realizează o structură redundantă de tip individual, fie la nivelul sistemului când se obține o structură redundantă de tip global. În Fig. 1.10 se prezintă modelul fizic al unui echipament electronic nerendundant format din *n* module funcționale $\prod_{i=1}^{n} M$ conectate serial:

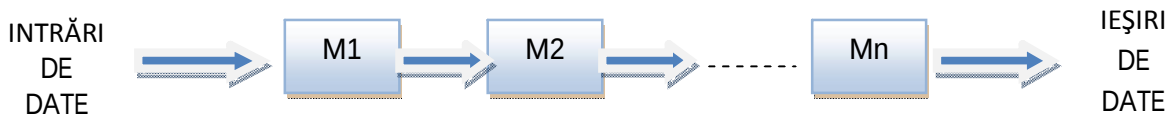
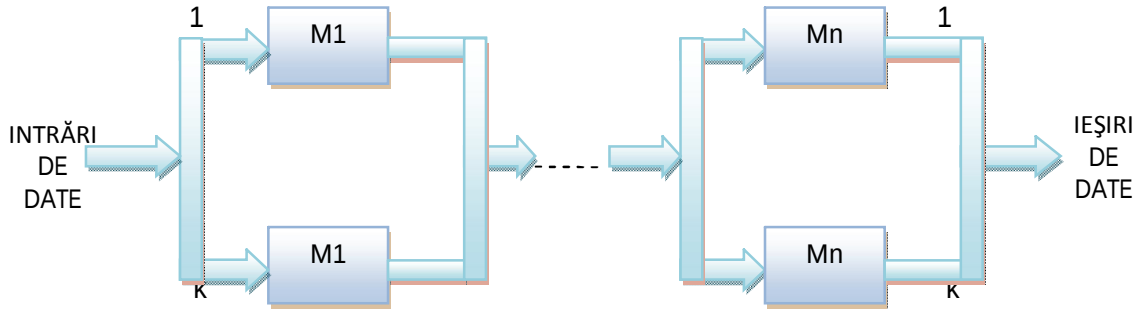


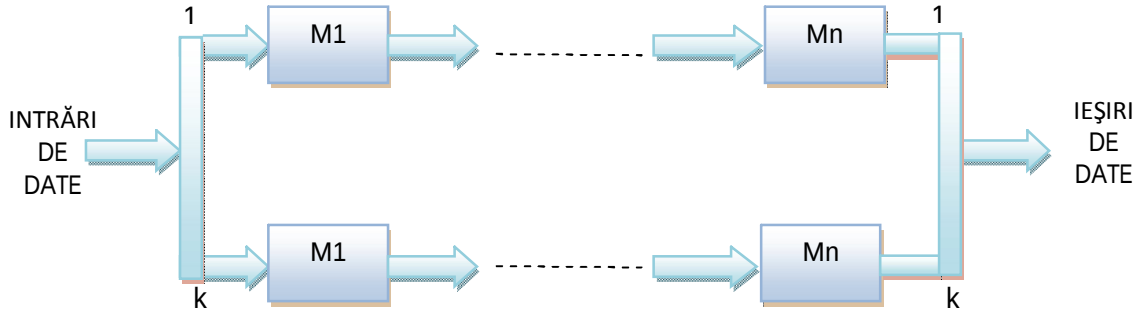
Fig. 1.10 Schema bloc a unui echipament electronic neredundant

Sistemul se află în stare de bună funcționare, atunci când funcționează corect toate modulele (funcționare serială), defectarea unui modul funcțional, atrăgând după sine întreruperea lanțului.

Aplicarea redundanței funcționale acestui model de echipament electronic, presupune multiplicarea modulelor funcționale şi conectarea lor aşa cum se prezintă în Fig. 1.11.



a) Redundanța individuală



b) Redundanța globală

Fig. 1.11 Schemele de implementare a redundanței protective statice prin procedee de multiplicare

Vom considera că modulele funcționale sunt nereparabile, defectările elementelor funcționale sunt independente sau staționare, iar circuitele de comutare a rezervelor sunt în stare de bună funcționare. Se pot scrie expresiile funcțiilor de fiabilitate pentru cele două structuri redundante analizate [HC-08b]:

- pentru structura redundantă individuală (Fig. 1.11 a)

$$R_I(t) = \prod_{i=1}^{n}\left[1-\left(1-R_i(t)\right)^{k_i}\right] \tag{1.4}$$

- respectiv pentru structura redundantă globală (Fig. 1.11 b)

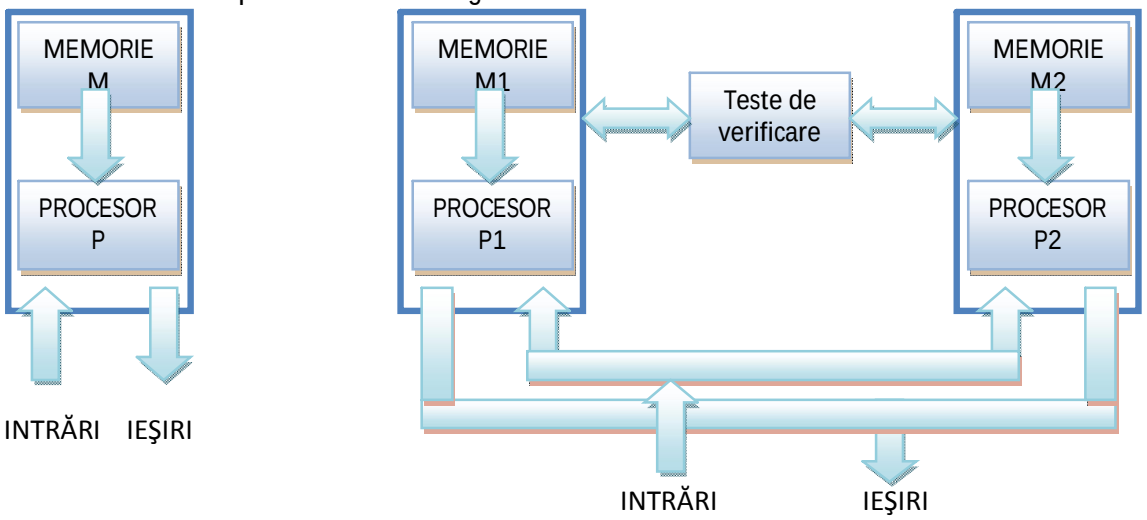
$$R_G(t) = 1-\left[1-\prod_{i=1}^{n}R_i(t)\right]^{k_i} \tag{1.5}$$

unde $R_I(t)$ reprezintă funcția de fiabilitate a modulului M_I din sistemul nerendundant iar R_i numărul de rezerve utilizate.

În Fig. 1.12 se prezintă configurația unui sistem de prelucrare numerică a informației cu structură redundantă, rezultată prin dublarea sistemului nerendundant. Ambele sisteme ale structurii funcționează "on *line*" şi execută sarcini identice cu scopul de a compara ieşirile, astfel încât sistemul se

va afla în stare de bună funcționare, chiar dacă unul dintre ele se va defecta. Sistemul conține şi circuite de verificare în timp real, care vor indica sistemul de procesare care funcționează eronat.

Trebuie remarcat că structurile redundante protective static rezultate prin multiplicare sunt recomandate să se utilizeze îndeosebi sistemelor analogice de prelucrare, întrucât ridică probleme dificile de sincronizare pentru sistemele digitale.

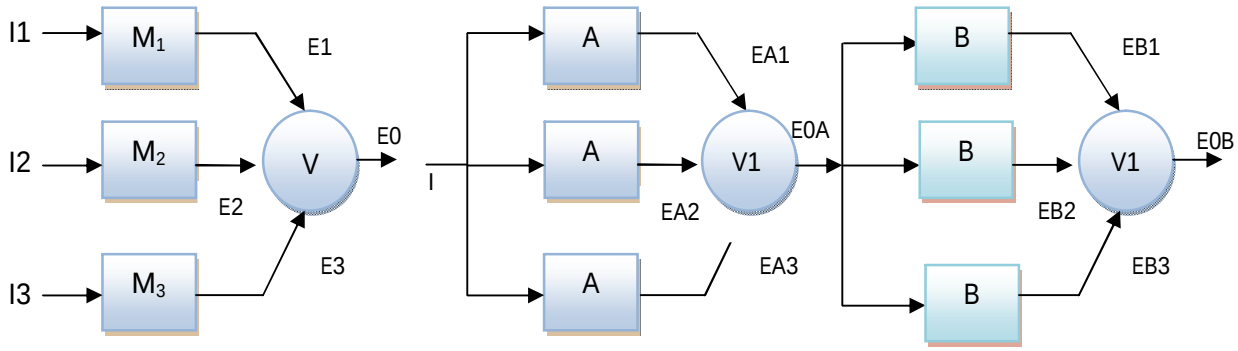


a) Sistem original *b) Sistem redundant*

Fig. 1.12 Aplicarea redundanței prin dublarea unui sistem de prelucrare a informației

1.4.3 Structuri redundante digitale cu logică majoritară

Structura redundantă digitală cu logică majoritară NMR (N-*modular redundancy)* este implementată prin divizarea sistemului neredundant în module funcționale, multiplicarea acestora de $N \in \mathbf{N}$ ori şi inserția între aceste module funcționale a unor sisteme de decizie denumite *"votere"* care funcționează după o logică majoritară. Cele mai cunoscute configurații [AB-05a] de structuri redundante cu logică majoritară sunt prezentate în Fig. 1.13(a – logică majoritară 2 din 3 TMR, respectiv b – logică majoritară n din 2n-1 NMR).



a) Logică majoritară 2 din 3 (TMR) b) Logică majoritară n din 2n-1 (NMR

Fig. 1.13 Implementarea structurilor redundante digitale cu logică majoritară

Modulele M_1, M_2, M_3, sunt identice din punct de vedere fizic şi funcțional, iar la intrările lor se aplică semnalele I_1, I_2, I_3, de asemenea identice. Sistemul de decizie(voterul *V)*, compară semnalele E_1, E_2, E_3, aplicate la intrarea sa. Dacă $E_1 = E_2 = E_3$, semnalul de ieşire E_0 va fi identic cu acestea. Dacă însă semnalul E_3 este incorect $E_1 = E_2 \neq E_3$, semnalul de ieşire va lua valoarea majorității:

$$E_0 = E_1 = E_2 \tag{1.6}$$

În acest mod structura redundantă TMR maschează defectarea unuia din modulele funcționale în care a fost partiționat sistemul.

Probabilitatea de bună funcționare a unui sistem cu structură redundantă digitală în logică majoritară de tip *n* din *2n-1*NMR poate fi scrisă[JW-05]:

$$R_R(t) = \left\{ \sum_{j=n}^{2n-1} C_{2n-1}^{j} R(t)^j [1 - R(t)]^{2n-1-j} \right\} R_v(t) \tag{1.7}$$

unde: $R(t)$ este funcția de fiabilitate a modulului funcțional iar $R_v(t)$ funcția de fiabilitate a voterului.

Un exemplu reprezentativ de implementare a structurii redundante cu logică majoritară o constituie sistemul de calcul autotestabil, cu structură tolerantă la defectări prezentat în Fig. 1.14.

Sistemul de decizie este divizat în două subsisteme: voterul de sincronizare şi voterul de date. Calculatorul master formează semnalul *"REQUEST"* după care datele pot fi citite de pe magistrala de date. Voterul de sincronizare generează semnalele de sincronizare pentru dialogul celor două sisteme de calcul *MASTER* şi *SLAVE,* astfel încât datele de operare să fie disponibile simultan la voterul de date conectat la alte magistrale [AF-03].

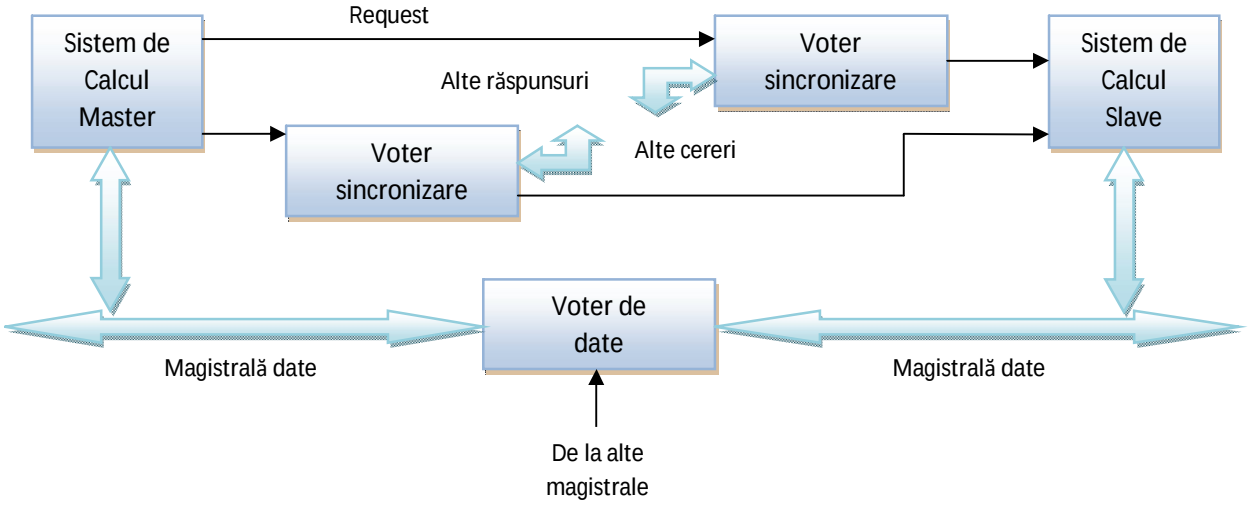


Fig 1.14. Arhitectura sistemului de calcul autotestabil cu structură tolerantă la defectări în logica majoritară

1.4.4. Structuri redundante protective cu logică cvadruplă

Această structură redundantă se aplică sistemelor numerice şi constă în multiplicarea datelor ori a circuitelor logice, conectate astfel încât, semnalele eronate sunt multiplexate cu semnalele corecte provenite de la circuitele de rezervă, realizând în acest fel mascarea defectelor la ieşirea sistemului.

Metoda se bazează pe caracteristicile unor circuite integrate logice de a masca intrinsec unele defectări de blocare pe "1" sau "0" a ieşirilor [AM-06].

Analizând circuitul din Fig. 1.15a se constată că o eroare la ieşirea porţii P_1 se va propaga la ieşirea porţii P_3 dacă combinaţiile logice ale semnalelor la intrările porţilor P_1 şi P_2 nu sunt identice.

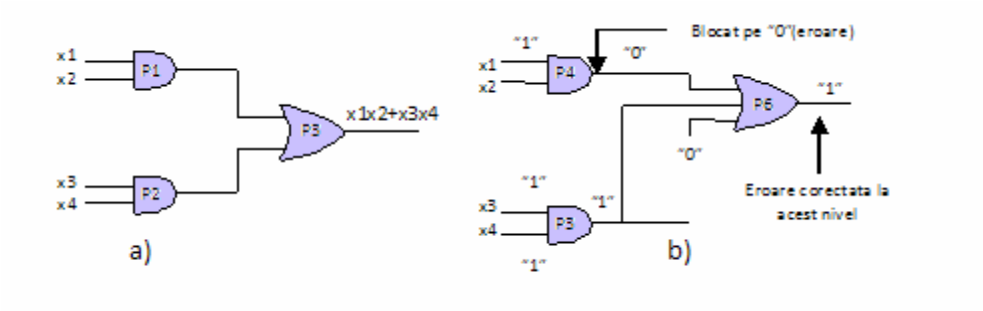


Fig. 1.15 Circuite logice de mascare a erorilor.

În caz contrar, poarta P_2 maschează defectarea poarta P_1, la ieşirea porţii P_3 semnalul de ieşire apărând corect. Pentru circuitul prezentat în Fig. 1.15b semnalele de ieşire ale porţilor logice redundante P_4, P_5 sunt aduse la intrarea porţii P_6 care va masca eroarea apărută la ieşirea porţii P_4.

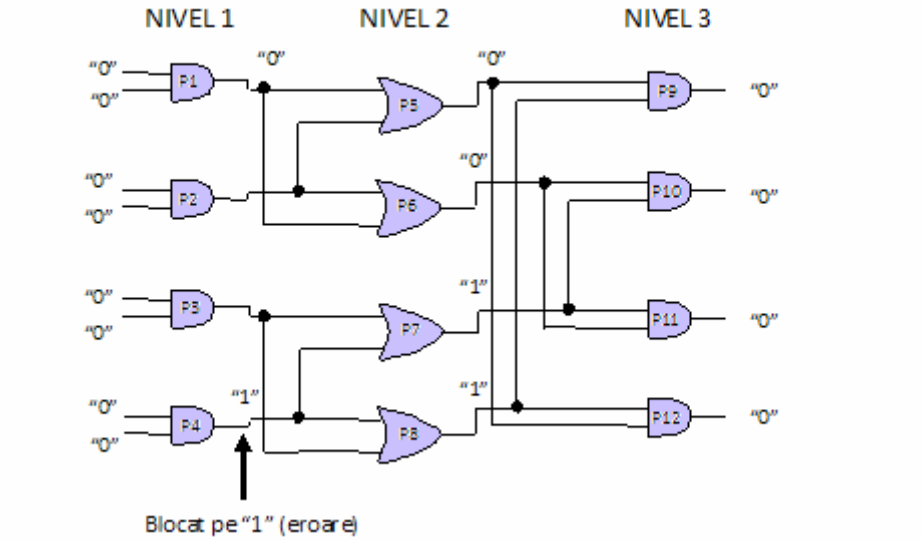


Fig. 1.16 Implementarea toleranţei la defecţiuni utilizând structura redundantă cu logică cvadruplă.

Orice eroare care apare la nivelul 1 se raportează instantaneu de logica circuitului la nivelul 2 pentru semnalele de "1" respectiv, la nivelul 3 pentru semnalele de "0" aplicate intrărilor circuitelor logice din primul nivel.

1.4.5. Structuri redundante protective prin codare

Codificând stările unui automat cu un cod redundant având biți de informație si control, se obține o structură redundantă protectivă prin codare [BD-02].

Implementarea redundanței prin codarea unui echipament electronic nu presupune multiplicarea completă a modulelor sau componentelor acestuia, dar implică utilizarea unor elemente de circuit adiționale, care permit reconstituirea semnalului de ieşire corect, chiar în prezența existenței unor defecțiuni. Să considerăm un automat finit de tip *Mealy* (Fig. 1.17) unde *M* reprezintă memoria sistemului; *S* şi *E* circuite combinaționale care realizează setul de funcții booleene y şi z.

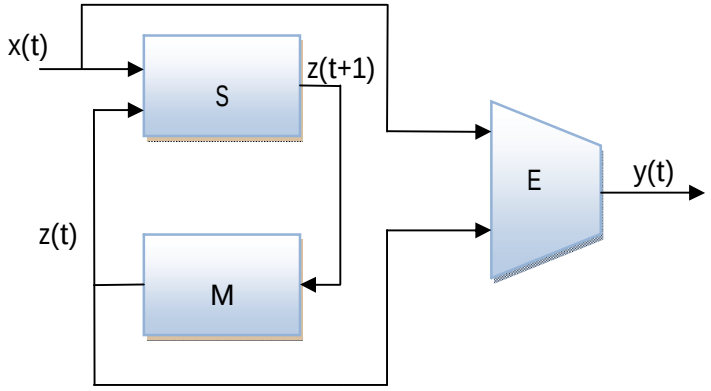


Fig. 1.17 Structura unui circuit automat finit tip Mealy

La momentul de timp t blocul *S* (SLAVE) al sistemului generează la ieşire starea următoare*: z(t+l)=z'(t)* ca o funcție de vectorul semnalelor de intrare *x(t)* şi vectorul stare al automatului *z(t).* Funcția de ieşire *y(t)* va fi generată de circuitul combinational *E* ca o funcție de vectorul semnalelor de intrare *x(t)* şi de vectorul de stare *z(t).*

$$\bigwedge_{j=1}^{m} y_j = y_j(x_1 \dots \dots x_k, z_1 \dots \dots z_k) \tag{1.8}$$

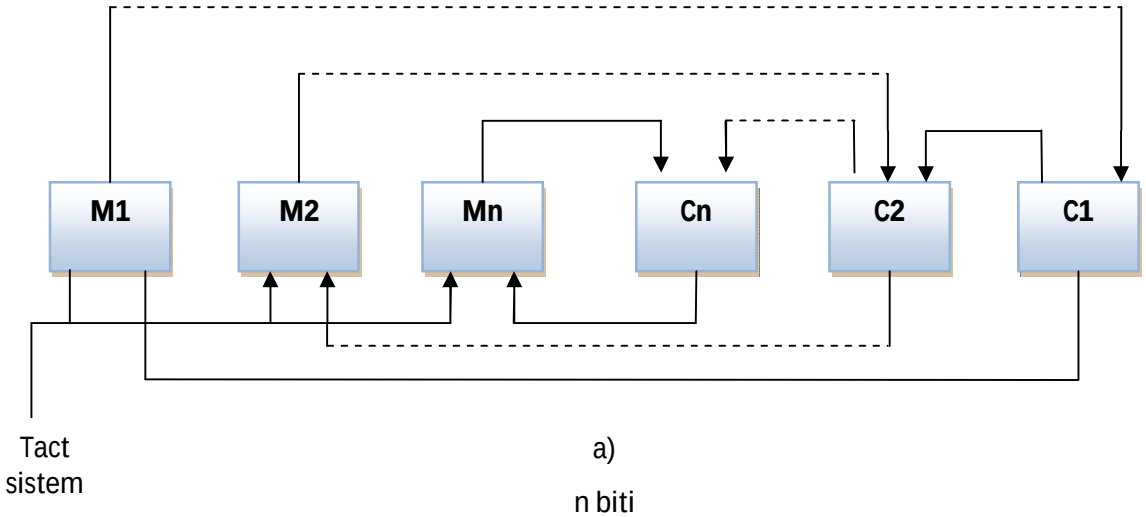
iar ecuațiile stării următoare vor fi de forma:

$$\bigwedge_{j=1}^{k} z_j = z_j(x_1 \dots \dots x_k, z_1 \dots \dots z_k) \tag{1.9}$$

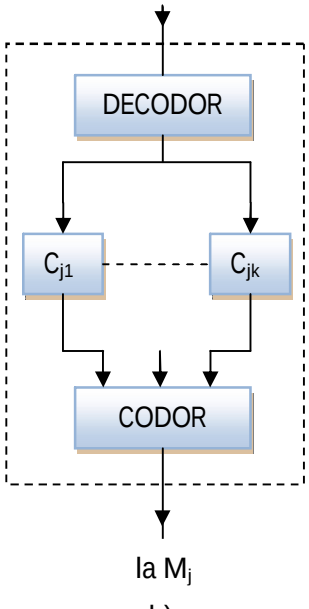
Acest model pune în evidență *2k* stări interne posibile ale automatului. Introducerea unor stări redundante pentru mascarea defectelor hard utilizând un cod corector de erori va conduce la creşterea numărului variabilelor de stare z şi implicit a numărului celulelor de memorie corespunzătoare.

Dacă numărul variabilelor de stare creşte de la *k* la n, atunci numărul de stări interne creşte de la 2^k la 2^n, astfel încât fiecărei stări a sistemului neredundant îi vor fi alocate 2^{n-k} stări. Atunci când se va defecta un element hard din subsistemele care realizează una dintre funcțiile de stare $z_j(t)$, sistemul astfel construit va masca defectul dacă toate cele 2^{n-k} stări alocate nu sunt afectate de acest defect [BM-04].

Fig. 1.18. prezintă o modalitate concretă de implementare a unei codări redundante *k* din *n* la un automat finit şi alăturat structura circuitelor de control. Pentru o aplicație concretă aceste circuite pot fi proiectate după principiile sistemelor secvențiale sincrone sau asincrone codarea şi decodarea realizându-se funcție de tipul codului corector de erori utilizat.



a)



b)

a) Automat cu structură redundantă utilizând coduri corectoare de erori
b) Structura cirbuitelor de control

Fig. 1.18 Implementarea unei codări redundante k din n unui automat secvențial finit

1.4.6. Structuri redundante protective dinamice şi hibride

Structura redundantă protectivă dinamică utilizează module identice din punct de vedere funcțional, o parte dintre ele fiind active pentru a realiza funcția de bază a sistemului, celelalte fiind rezerve ce urmează a fi conectate, numai atunci, când unul dintre modulele active se defectează (de aici şi denumirea de structură redundantă de comutație).

Prin acest mod de înlocuire automată a modulului defect cu un modul de rezervă în stare de bună funcționare, structura redundantă protectivă dinamică realizează funcția unui sistem autoreparabil [CL-00a].

Fig.1.19 prezintă schema simplificată a unui sistem de calcul cu structură redundantă protectivă de comutație în configurație de sistem autoreparabil.

Sistemul A_1-B_1 funcționează *on-line*, iar sistemul A_2-B_2 funcționează *off-line* şi reprezintă rezerva care se conectează doar în momentul defectării sistemului de bază. Deşi softul şi hardul sunt deosebit de complexe pentru structurile redundante dinamice, performanțele de fiabilitate sunt remarcabile.

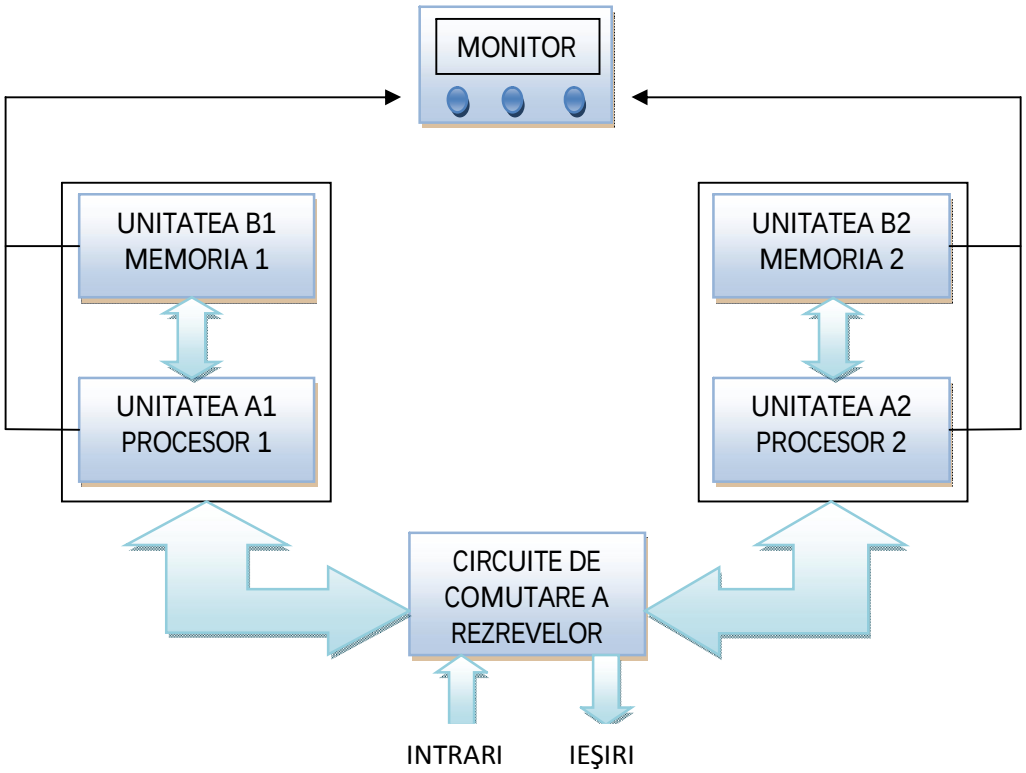


Fig. 1.19 Sistem de calcul cu structură redundantă protectivă dinamică

Structura redundantă hibridă este o combinație între caracteristicile structurilor redundante cu logică majoritară şi de comutație, ceea ce justifică interesul manifestat pentru utilizarea sa în scopul implementării toleranței la defecțiuni. Schema tipică de realizare a acestei structuri este prezentată în Fig.1.20.

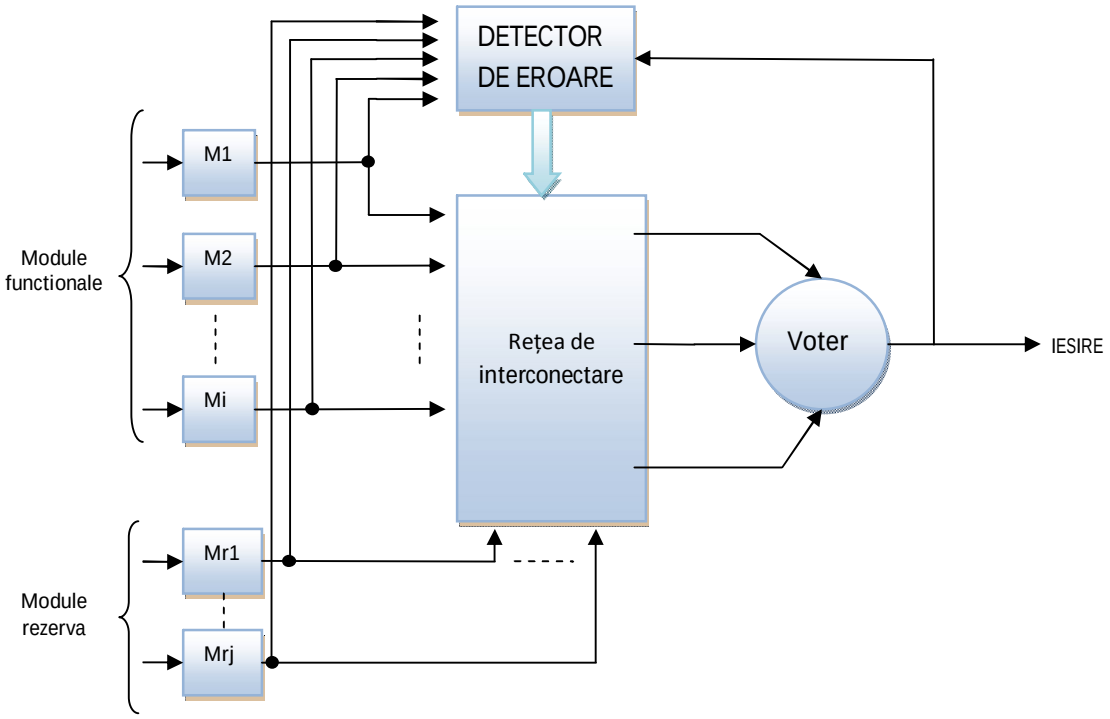


Fig. 1.20 Schema structurii redundante hibride

Schema are la bază un nucleu de *2n+1* module funcționale identice conectate prin intermediul rețelei de interconectare pentru a forma o structură cu logică majoritară de tipul *n+1* din *2n+1* şi un număr de *r* module de rezervă care urmează să fie conectate numai în momentul detectării unor defecțiuni la oricare din cele 2n+*1* module funcționale.

Detectorul de eroare testează periodic ieşirile modulelor nucleului şi pune în evidență defectarea unui modul. Rețeaua de interconectare este astfel concepută încât să treacă la înlocuirea automată a modulului defect cu o rezervă, atunci când constată că una din ieşirile testate sunt eronate.

O astfel de schemă prezintă în general o fiabilitate foarte ridicată dacă voterul, sistemul de comutație şi detectorul de eroare sunt concepute într-o tehnologie avansată.

1.5. STRUCTURI REDUNDANTE DE INTERCONECTARE

La realizarea unui sistem electronic complex de înaltă fiabilitate cu structură tolerantă la defecțiuni, devine necesar ca şi interconexiunile dintre elementele sistemului să admită toleranță la defectările posibile. Astfel pentru toate formele de comunicații digitale se disting trei tipuri de structuri pentru bus-urile de date diferite dar complementare:

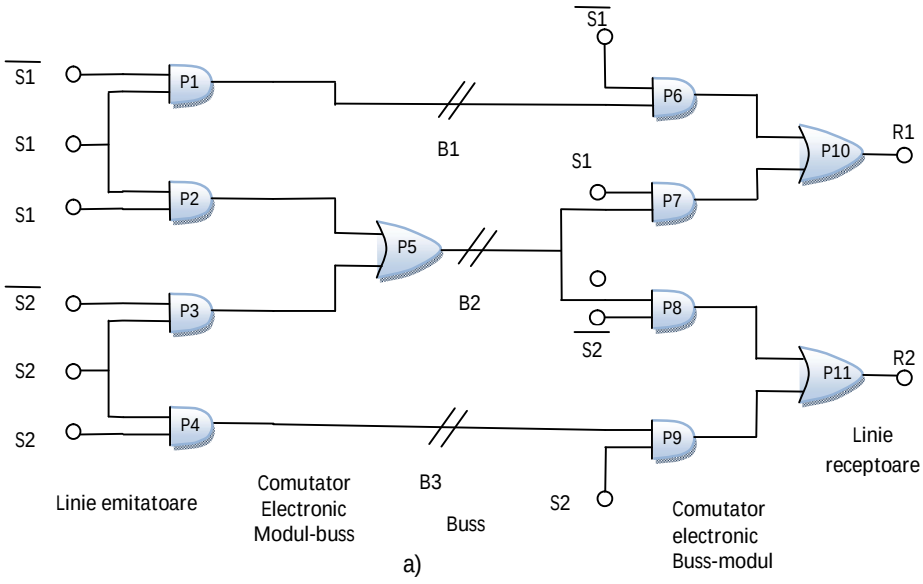
- *"structuri redundante de tip static"* care utilizează codurile corectoare de erori transmise pe linii de date redundante;
- *"structuri redundante de tip dinamic"* care implică utilizarea mecanismelor de detecție a defectărilor pe linie şi înlocuirea cu linii de rezervă obținând astfel autorepararea sau reconfigurarea pentru tolerarea defecțiunilor;

- *"structuri redundante distribuite"* care asigură o redundanță topologică intrinsecă magistralelor de date şi tolerarea defectărilor prin utilizarea unor rute alternative. Conduce însă la o limitare a performanțelor de operare ale sistemului.

1.5.1. Structura redundantă dinamică aplicată magistalelor de date

O abordare posibilă [DA-05] pentru implementarea redundanței dinamice la bus-urile de date este considerarea celor l linii de informație ca l module funcționale cărora li se aplică tehnicile clasice de redundanță prezentate în Paragraful 1.2. Această abordare devine nerealistă, datorită numărului mare de linii de informație solicitate în practică. Dacă însă se are în vedere, că cele l linii de informație se conectează la magistralele de date B_j, $B_{j+1}...B_{j+k}$ va rezulta un sistem de comunicație mai simplu. Numărul maxim de stări posibile al liniilor de transmisiune este $k+1$, dar cerințele de *Fan-in* şi *Fan-out* se reduc de la valoarea l la valoarea k, care de obicei este cu un ordin de mărime mai mic.

Considerăm că informația se transmite de la un modul emițător pe două linii S_1, S_2 către două linii ale modulului receptor R_1, R_2 prezentat în Fig. 1.21. S-a prevăzut o linie de rezervă $(k=1)$ astfel încât fiecare linie de informație să admită două stări care vor fi memorate în registrul stărilor BSR. Comutatorul electronic va comuta linia de informația S_1 la buss-ul 1 sau 2 iar linia de informație S_2 la buss-ul 2 sau 3 funcție de starea defectului detectat.



STARI		Linie buss neconectata
S1	S2	
0	0	B3
0	1	B2
1	1	B1

b)

a) Schema electronică *b) Starile buss-urilor*

Fig. 1.21 Structura redundantă de comutație aplicată buss-urilor de date

Se constată că fiecare linie de informație poate fi conectată în acelaşi timp la două linii de buss. Dacă celula BSR_j se află în starea logică "0" linia de informație este conectată la linia de buss superioară, iar dacă celula se află în starea logică "1" linia se conectează la linia de buss următoare. Desigur că un defect al registrelor BSR poate conduce la defectarea sistemului. O soluție [DC-05b] de aplicare a redundanței pentru registrele BSR, o constituie triplarea lor în funcționare paralelă, iar comutatorul electronic se înlocuieşte cu un aranjament de votere cu trei grade de libertate.

În continuare vom estima complexitatea regiştrilor BSR a comutatorului modul-buss (nBS) şi a comutatorului buss-modul (BMS). Sistemul va conține *l* registre BSR cu $log_2(1 + k/q)$ celule bistabile fiecare, unde *k* reprezintă numărul de rezerve, iar *q* este un divizor comun al parametrilor *l şi* k.

Dacă notăm cu *Q* complexitatea sistemului, rezultă pentru registrul BSR:

$$Q_{BSR} = k \cdot l \left[log_2 \left(1 + \frac{k}{q} \right) \right] \tag{1.10}$$

Având în vedere că fiecare grup independent al sistemului are *l/q* şi *k/q* rezerve, complexitatea comutatorului MBS va fi:

$$Q_{MBS} = 2l + r + \frac{lk}{q} - 2q \tag{1.11}$$

În mod similar se stabileşte complexitatea comutatorului BMS

$$Q_{MBS} = 2l + \frac{lk}{q} \tag{1.12}$$

Dacă considerăm schema de interconectare la magistrale de date, ca un sistem format din $m=l/q$ elemente şi $n=(l-k)/q$ grupe de linii, se obține funcția de fiabilitate a unei grupe:

$$R(t) = \sum_{j=0}^{m-n} C_n^j R(t)^{n-j} [1 - R(t)]^j \tag{1.13}$$

iar pentru cele *q* grupe ale sistemului de interconectare, funcția de fiabilitate va avea expresia:

$$R_{SIST}(t) = \left\{ \sum_{j=0}^{k l_q} C_{\frac{l+k}{q}}^j R(t)^{\frac{l+k}{q}-j} \cdot [1 - R(t)]^{\frac{j}{l}} \right\}^k \tag{1.14}$$

Putem concluziona acum că alegerea numărului de conexiuni de rezervă *k*, a numărului de grupe *q*, pentru un număr dat de linii de informație *l*, ca şi valoarea funcției de fiabilitate pentru o linie de buss *R(t)*, este influențată de următorii trei factori:

a) valoarea funcției de fiabilitate pentru sistemul de interconectare;

b) creşterea complexității care trebuie să fie rezonabilă;

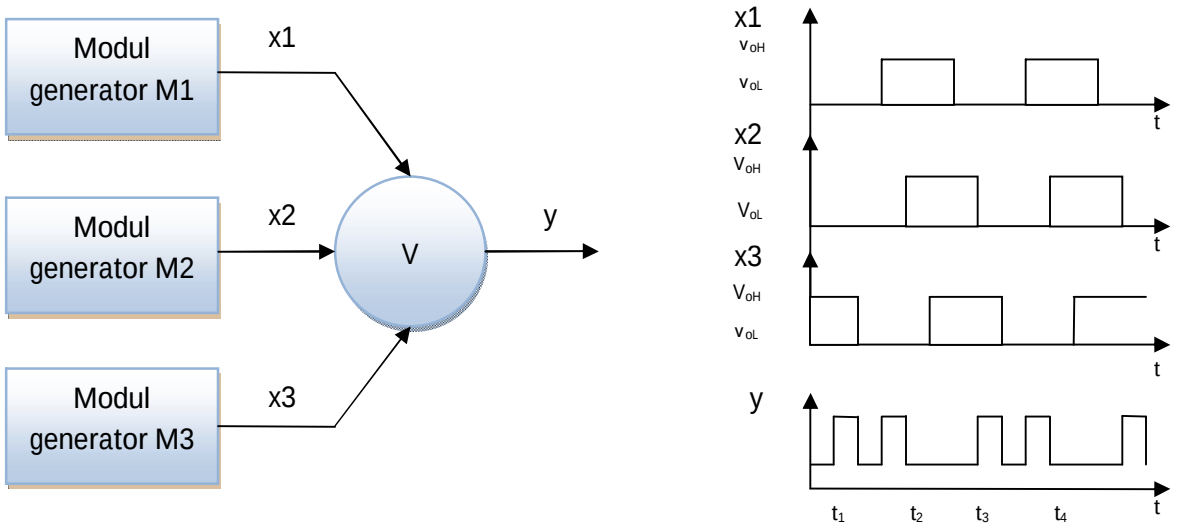
c) limitarea numărului de conexiuni intrare-ieşire acceptat.

Alte structuri tolerante la defecțiuni aplicate magistralelor de date pot fi: de tip *"gracefully degrading"* sau structuri reconfigurabile, pentru care bussul îşi modifică capacitatea la apariția unor defecțiuni (de exemplu de la 32 la 16 biți).

1.5.2 Sincronizarea sistemelor digitale tolerante la defecțiuni

Implementarea structurilor redundante protective pentru sistemele logice impun o serie de restricții legate de sincronizarea funcționării unităților constituente. Analiza poate fi efectuată cauzal pentru fiecare structură redundantă protectivă analizată în paragraful 1.4. În cadrul acestui paragraf vom face un studiu concret asupra sincronizării echipamentelor electronice cărora li s-a implementat structură redundantă cu logică majoritară care este mai des utilizată în practică[ZW-05].

Considerând cazul structurii redundante cu logică majoritară de tipul 2 din 3, sincronizarea ca şi întârzierile introduse de către cele trei module funcționale implică introducerea de erori în funcția de transfer pe intervalul de timp t_2-t_1 respectiv t_4-t_3 aşa cum se exemplifică în Fig. 1.22.



a) b)

a) Schema generatorului de impulsuri cu structură redundantă în logica majoritară 2 din 3
b) Formele de undă la intrări şi la ieşiri

Fig. 1.22 . Defecte de sincronizare în funcționarea sistemelor redundante cu logică majoritară de tip 2 din 3.

Semnalele x_1 x_2, x_3 reprezintă formele de undă generate de cele trei module funcționale ale structurii, iar *y* semnalul de la ieşirea sistemului de decizie. Pe duratele de timp t_2-t_3 respectiv t_4-t_3 se constată că semnalul *y* depinde de semnalul întârziat defazat x_3.

Intervalul de timp de întârziere a funcției de transfer cu logică majoritară este:

$$\Delta t_{max} = \underset{i=1}{\overset{n}{Max}} \left(t_{\Delta M_i M_{i+1}} \right) \tag{1.15}$$

unde $t_{\Delta M_i M_{i+1}}$, reprezintă timpul de propagare (întârziere) între modulele M_i, respectiv M_{i+1}.

Dacă răspunsul voterului apare cu o întârziere *to* după prima tranziție şi satisface inegalitățile:

$$\Delta t_{max} \ll o \ll \frac{1}{2} - t_{\Delta M_i M_{i+1}} \tag{1.16}$$

Δt_{max} reprezentând perioada tactului sistemului, nesincronizările nu produc efecte catastrofale (erori in funcționare).

Trebuie subliniat că sincronizarea funcționării modulelor multiplicate ale unei structuri redundante protective cu logică majoritară, reprezintă o problemă deosebit de importantă la implementarea toleranței la defecțiuni, sistemelor digitale de prelucrare. Din acest motiv, practic această sincronizare se asigură de către un semnal extern care prescrie momentele de validare a informației la ieşirile modulelor active [FE-05].

1.5.3 Criterii de comparare a performanțelor sistemelor digitale tolerante la defecțiuni

Cea mai uzuală metodă de comparare a performanțelor structurilor redundante protective, aplicată echipamentelor electronice, o reprezintă analiza indicelui logaritmic de îmbunătățire a fiabilității, dar şi a indicilor costului şi eficienței pentru sistemul cu structură redundantă, comparativ cu sistemul având structură neredundantă [HC-07].

În continuare vom prezenta cele mai uzuale exprimări folosite în literatura de specialitate pentru definirea indicelui de creştere a fiabilității.

- Indicele de îmbunătățire a fiabilității sistemului I_R exprimat ca raport al probabilităților de bună funcționare pentru sistemul cu structură redundantă şi neredundantă:

$$I_R = \left(\frac{R_r}{R_n} \right)^k \tag{1.17}$$

unde k reprezintă numărul subsistemelor în care a fost partajat sistemul în vederea aplicării redundanței protective.

- Indicele de îmbunătățire a fiabilității sistemului I_m exprimat ca raport al timpilor medii de bună funcționare pentru sistemul redundant m_R respectiv neredundant m_N.

$$I_m = \frac{m_R}{m_N} \tag{1.18}$$

unde *m* reprezintă media timpului de bună funcționare.

Întrucât:

$$m = \int_0^{\infty} t \frac{dR}{dt} dt$$

(1.19)

se obține:

$$I_m = \frac{\int_0^{\infty} t \frac{dR_R}{dt} dt}{\int_0^{\infty} t \frac{dR_N}{dt} dt} = \frac{\int_0^{\infty} R_R^{k-1} \frac{dR_{rR}}{dt} dt}{\int_0^{\infty} R_N^{k-1} \frac{dR_N}{dt} dt} \quad (1.20)$$

- Indicele de îmbunătățire a fiabilității sistemului I_z exprimat ca raportul ratelor de defectare pentru sistemul cu structură neredundantă z_N, respectiv redundantă z_R, definit cu relația:

$$I_z = \frac{z_N(t)}{z_R(t)} \quad (1.21)$$

Întrucât:

$$z(t) = -\frac{1}{R(t)} \cdot \frac{dR(t)}{dt} \quad (1.22)$$

ecuația (1.21) devine:

$$I_z = \frac{R_R \frac{dR_N}{dt}}{R_N \frac{dR_R}{dt}} \quad (1.23)$$

- Indicele logaritmic de îmbunătățire a fiabilității sistemului I_{log} cu structură redundantă este definit cu relația:

$$I_{log} = \alpha = \frac{lnR_N(t)}{lnR_R(t)} \quad (1.24)$$

Dezvoltând in serie logaritmul rezultă:

$$\alpha = \frac{ln(1-F_N)}{ln(1-F_R)} = \frac{F_N}{F_R}\left(1 + \frac{F_N - F_R}{2}\right) \quad (1.25)$$

Se constată că numai indicele α este independent de complexitatea sistemului si este sensibil la variațiile unice ale valorii funcției de fiabilitate, ceeace îl recomandă a fi utilizat la analiza fiabilității sistemelor cărora li s-a implementat redundanța protectivă, la defecțiuni.

- Indicele de cost:

$$I_C = \frac{costul\ sistemului\ redundant}{costul\ sistemului\ neredundant} \tag{1.26}$$

Respectiv eficiența utilizării tehnicilor redundante:

$$E = \frac{indicele\ logaritmic\ de\ creştere\ a\ fiabilităţii}{indicele\ de\ cost} \tag{1.27}$$

Desigur utilitatea practică a indicilor definiți anterior, se poate verifica pentru o analiză concretă comparativă a diferitelor tehnici de implementare a redundanței protective, în vederea realizării structurilor autotestabile tolerante la defecțiuni.

1.6 IMPLEMENTAREA STRUCTURILOR REDUNDANTE LA NIVEL OPTIMAL

Cercetările actuale cu privire la variația costului unui sistem redundant, aplicat în vederea asigurării toleranței la defecțiuni, au evidențiat o scădere logaritmica a probabilității de defectare a sistemului odată cu creşterea costului, datorită aplicării redundanței protective. De aceea o problemă importantă care trebuie soluționată în cazul proiectării sistemelor cu structură redundantă, o constituie determinarea gradului optim al redundanței, precum şi precizarea nivelului la care trebuie aplicată redundanța, astfel încât fiabilitatea sistemului în prezența diferitelor restricții sa fie maximă.

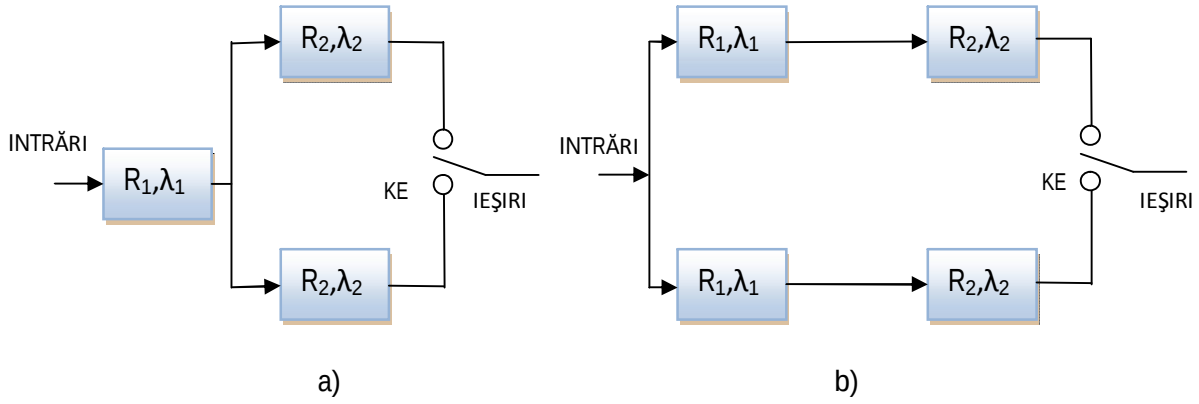
1.6.1 Strategii de implementare a redundanței

Vom încerca să răspundem la întrebarea? *"pentru a obține creştere a funcției de fiabilitate, mărim gradul redundanței prin alocarea unei rezerve suplimentare, sau partiționăm în mod optimal sistemul, aplicându-i acelaşi tip de redundanță de acelaşi grad, la alt nivel"*

Următoarele două exemple evidențiază importanța unei analize apriorice a tehnicilor de implementare a redundanței şi prin aceasta vom putea adopta strategia cea mai eficientă pentru o aplicație concretă.

Exemplu 1.6.1

Se consideră sistemul cu structură redundantă dinamică de ordinul 1 din Fig. 1.23.



a) Structura redundantă este aplicată numai elementului 2 din sistem

b) Structura redundantă este aplicată întregului sistem

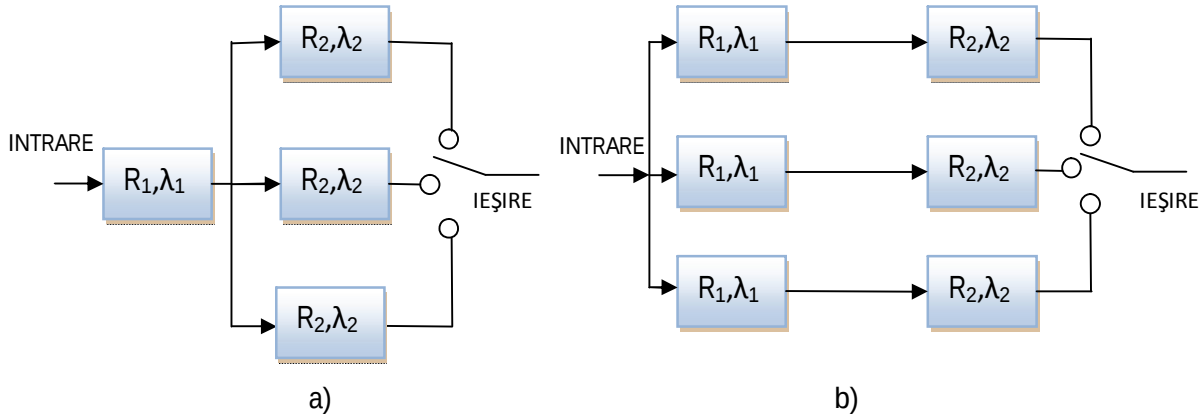
Fig. 1.23 Structură redundantă dinamică de ordinul 1 implementată unui sistem

Presupunem $R_1(t)>R_2(t)$ şi notăm cu $R_a(t)$ respectiv $R_b(t)$ funcțiile de fiabilitate ale schemelor din fig.2.18 a) respectiv fig.2.18 b). Calculând:

$$R_b(t)-R_a(t)=2R_1(t)\cdot R_2(t)-R_1(t)\cdot 2R_2(t)^2-2R_1(t)\cdot R_2(t)+R_1(t)\cdot R_2(t)^2=R_1(t)\cdot R_2(t)^2\cdot[1-R_1(t)]>0\forall R_1(t),R_2(t)\in[0,1] \quad (1.28)$$

adică sistemul din Fig. 1.23b este mai fiabil decât sistemul din Fig. 1.23a.

La o concluzie identică se ajunge dacă gradul redundanței este crescut la 2 aşa cum se arată în Fig. 1.24.



a) Structura redundantă este aplicată numai elementului 2 din sistem

b) Structura redundantă este aplicată întregului sistem

Fig. 1.24 Structură redundantă dinamică de ordinul 2 implementată unui sistem

Într-adevăr: R_b-R_a>0 atunci când:

$$3 - R_2(t) - 3R_1(t) + R_1(t)^2 \cdot R_2(t) > 0 \tag{1.29}$$

ceeace este adevărat, deoarece:

$$R_2(t) < \frac{3}{R_1(t)} + 1 \qquad \forall R_1(t) \in [0,1] \tag{1.30}$$

Ca o concluzie finală rezultă că alocarea unei rezerve pentru sistemul 1 va conduce implicit la creşterea fiabilităţii întregului sistem.

Exemplu 1.6.2

Vom considera pentru analiza aceleaşi structuri redundante prezentate în Fig. 1.23 şi Fig. 1.24. Notăm sistemul din Fig. 1.23b cu S_1 iar cel din Fig. 1.24a cu S_2. Întrucât nu putem decide aprioric asupra funcţiei de fiabilitate a celor două structuri vom calcula:

$$R_{S_2} - R_{S_1} = R_1(t)\left[R_2(t)^3 - 3R_2(t) + 2R_2(t) + R_1(t)R_2(t)^2\right] = \tag{1.31}$$
$$= R_1(t) \cdot R_2(t)\left[R_2(t) - 3R_2(t) + R_1(t) \cdot R_2(t) + 1\right]$$

Semnul acestei diferenţe se poate studia analizând reprezentarea grafică a funcţiei:

$$R_1(t) = \frac{3R_2(t) - R_2^2(t) - 1}{R_2(t)} = f(R_2(t)) \tag{1.32}$$

în planul $R_1(t)$, $R_2(t)$ (vezi Fig. 1.25)

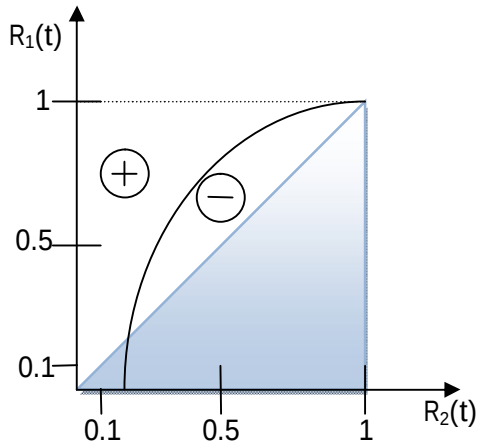


Fig. 1.25. Reprezentarea grafică a funcţei de fiabilitate pentru cele două sisteme analizate

Din analiza graficului se pot stabili condiţiile pentru care sunt satisfăcute inegalităţile $R_{S_2}(t) \gtrless R_{S_2}(t)$. Zona notată cu $\ominus$ indica situaţiile pentru care sistemul S_2, cu toate că este mai scump

decât S_1, este mai puțin fiabil, iar zona ⊕ stabileşte condițiile pentru care sistemul S_2 este mai fiabil decât sistemul S_1.

Creşterea fiabilității sistemului al cărui model structural de fiabilitate este prezentat în Fig. 1.26a poate fi rezolvată în două moduri:

- adoptând o altă modalitate de partiționare a sistemului şi aplicându-i acestuia, aceeaşi structură redundantă (strategia I reprezentată în Fig. 1.26b);
- fie creşterea gradului redundanței, prin alocarea unui element de rezervă suplimentar (strategia II reprezentată în Fig. 1.26c)

Alegerea uneia dinte cel două strategii se poate face prin compararea rapoartelor dintre variația funcției de fiabilitate şi variația costului. Vom presupune că sistemul are timpul de funcționare repartizat după o lege exponențială. În cazul strategiei 1 de implementare a redundanței s-a considerat sistemul partiționat în două subsisteme cu ratele de defectare λ_1, respectiv λ_2 astfel încât:

$$\lambda_1 + \lambda_2 = \lambda_m \tag{1.33}$$

unde: λ_m este rata de defectare a sistemului neredundant. Cu aceste date se poate scrie expresia funcției de fiabilitate a sistemului cu structură redundantă din Fig. 1.26b.

$$R_{SR}(t) = R_1(t) \cdot R_2(t)[2 - R_2(t)] = e^{-\lambda_m t}\left[2 - e^{(\lambda_m + \lambda_2)t}\right] \tag{1.34}$$

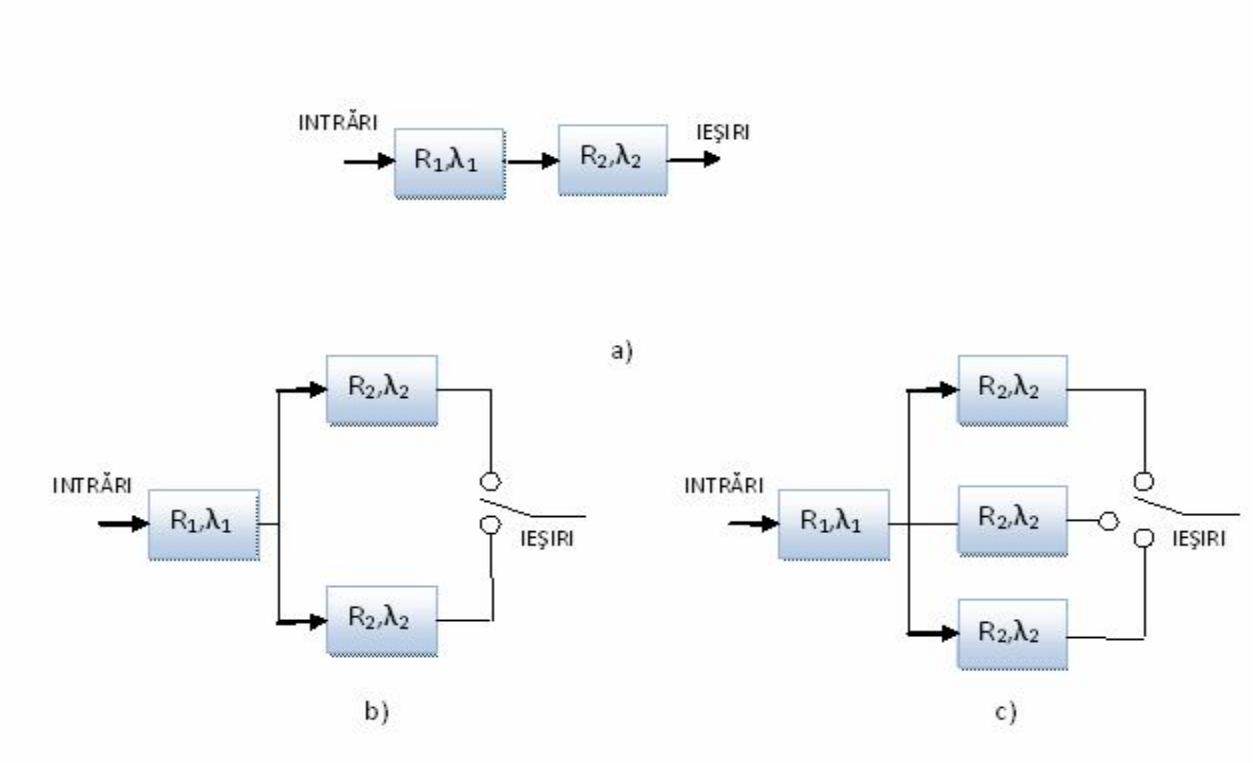


Fig. 1.26 Strategii de creştere a fiabilității unui sistem

Costul *c* alocat sistemului cu structură redundantă considerat, depinde de timpul misiunii *T şi* de rata de defectare λ_1 şi λ_2 după legea:

$$c = k(\lambda_1 + \lambda_2)T \tag{1.35}$$

Funcția de fiabilitate pe durata *T* a misiunii este:

$$R_{SR}(T) = 2e^{-\lambda_n T} - e^{-(2\lambda_n - \lambda_1)T} = k' - exp\left(\frac{c}{k}\right) \tag{1.36}$$

unde: *k'*şi *k* sunt constante independente de variația costului. Pentru acest caz variația funcției de fiabilitate cu modificarea costului este:

$$\frac{dR_{SR}(c,T)}{dc} = \frac{1}{k} \cdot e^{-(\lambda_1 - 2\lambda_2)T} \tag{1.37}$$

O analiză similară aplicată sistemului cu structură redundantă din Fig.1.26c arată că:

$$R_{SR}(t) = e^{-\lambda_1 T}\left[1 - \left(1 - e^{-\lambda_2 T}\right)^n\right] \tag{1.38}$$

(1.39)

(1.40)

$$\frac{dR_{SR}(T,n)}{dn} = e^{-\lambda_1 T}\left(1 - e^{-\lambda_2 T}\right)^n \cdot ln\left(1 - e^{-\lambda_2 T}\right) \tag{1.41}$$

$$\frac{dn}{dc} = \frac{1}{k\lambda_2 T}$$

$$\frac{dR_{SR}(C)}{dc} = \frac{e^{-\lambda_1 T}}{k\lambda_2 T}(1 - e^{-\lambda_2 T})^n \cdot ln(1 - e^{-\lambda_2 T}) \tag{1.42}$$

Pentru a facilita alegerea stategiei optimale de aplicare a redundanței protective, se determină raportul variației funcției de fiabilitate cu costul solicitat pentru cele două strategii:

$$k = \frac{\left.\frac{dR_{SR}(c,T)}{dc}\right|_{SII}}{\left.\frac{dR_{SR}(c,T)}{dc}\right|_{SI}} = -\frac{(1 - e^{-\lambda_2 T})^3 \cdot ln(1 - e^{-\lambda_2 T})}{\lambda_2 T \cdot e^{-2\lambda_2 T}} \tag{1.43}$$

Atunci când raportul k>1 este preferabilă aplicarea strategiei II de alocare a unei rezerve suplimentare, rezultând o creştere a fiabilității la aceeaşi variație a costului. Când raportul k<I este recomandată aplicarea strategiei I de modificare a partiționării sistemului, întrucât în acest caz se produce o creştere mai mare a funcției de fiabilitate la variația costului.

1.6.2. Metodă de stabilire a nivelului optim de partiționare a unui sistem, pentru aplicarea redundanței protective

Vom prezenta în continuare o metodă de identificare a nivelului optim de partiționate a unui sistem, astfel încât să se obțină o creştere a fiabilității la valoarea maximă pentru acelaşi grad de aplicare a redundanței protective [HC-02].

Analizăm cazul unui sistem având structură dinamică de timp având:

- *R(t)* funcția de fiabilitate a comutatorului electronic;
- $R(t)^N$ funcția de fiabilitate a sistemului căruia i s-a aplicat redundanța prootectivă.
- N-raportul dintre numărul circuitelor electronice echivalente din punct de vedere al funcției de fiabilitate a sistemului neredundant şi redundant.

Partiționăm sistemul în *k* subsisteme, cărora li se aplică tehnica de redundanță considerată. Vom demonstra că există o valoare unică a lui *k*, astfel încât să se obțină pentru sistemul cu structură redundantă o funcție de fiabilitate cu valoarea maximă. Funcția de fiabilitate pentru sistemul redundant este:

$$R_{SR}(t) = R(t)^N \left[R(t)\left(2 - R(t)\right)^{\frac{n}{k}}\right]^k \tag{1.44}$$

Vom proceda la analiza semnului derivatei funcției $R_{SR}(t)$ in raport cu *k:*

$$\frac{\partial R_{SR}(t,k)}{\partial k} = R(t)^n \cdot \frac{\partial}{\partial k}\left\{\left[R(t)\left(2 - R(t)\right)^{\frac{N}{k}}\right]^k\right\} \tag{1.44}$$

$$sgn\frac{\partial R_{SR}(t,k)}{\partial k} = sgn\frac{\partial}{\partial k}\left\{\left[R(t)\left(2 - R(t)\right)^{\frac{N}{k}}\right]^k\right\} = sgn\frac{\partial}{\partial k}\left\{\left[\rho(R)^k\right]\right\} \tag{1.45}$$

Dar:

$$\frac{\partial}{\partial k}\left[\rho(R)^k\right] = R\rho'(R)\left[\rho(R)\right]^{k-1} + \left[\rho(R)\right]^k \cdot \ln\rho(k) \tag{1.46}$$

Întrucât $\left[\rho(R)\right]^k > 0 \quad \forall k \in \mathbf{R}$ rezultă:

$$sgn\frac{\partial}{\partial k}\left\{\left[R(t)\left(2 - R(t)\right)^{\frac{N}{k}}\right]^k\right\} = sgn\left\{\begin{array}{c} R(t)^{\frac{N}{k}} \cdot \frac{N}{k}\ln R(t) + \\ \left[2 - R(t)^{\frac{N}{k}}\right] \cdot ln\left[R(t)\left[2 - R(t)^{\frac{N}{k}}\right]\right] \end{array}\right\} \tag{1.47}$$

Pentru gradul maxim de partiționare al sistemului *k=N*, semnul derivatei este:

$$\begin{aligned} sgn\frac{\partial R_{SR}(t,k)}{\partial k} &= sgn\{R(t)\,ln\,R(t) + [2 - R(t)]\,ln[R(t)(2 - R(t))]\} = \\ &= sgn\{2\,ln\,R(t) + [2\,ln\,R(t) + [2 - R(t)]]\,ln[R(t)(2 - R(t))]\} \end{aligned} \tag{1.48}$$

Problema s-a redus astfel la studierea semnului funcției:

$$\Psi(t) = 2lnR(t) + [2 - R(t)]ln[2 - R(t)] \tag{1.49}$$

Se constată că $\frac{d\Psi(t)}{dR(t)} > 0$ pentru *0<R(t)<1* deci funcția este continuă pe domeniul de definiție şi creşte o dată cu creşterea R(t). Rezultă ca $sgn\left.\frac{\partial R_{SR}(t,k)}{\partial k}\right|_{k=k_0} < 0$.

Pentru *k=1* respectiv pentru gradul minim de partiționare al sistemului:

$$sgn\left.\frac{\partial R_{SR}(t,k)}{\partial k}\right|_{k=1} = sgn\{2ln[2R(t)]\} > 0 \tag{1.50}$$

Deoarece: $\frac{\partial R_{SR}(t,k)}{\partial k}$ este o funcție continuă în raport cu $R \in [1, N]$, rezultă că există cel puțin un k_0 pentru care:

$$\left.\frac{\partial R_{SR}(t,k)}{\partial k}\right|_{k=0} = 0 \tag{1.51}$$

atunci când. Deci, funcția de fiabilitate a sistemului redundant considerat va avea un maxim la variația partiționații sistemului.

În continuare vom demonstra că această valoare este unică. Calculăm:

$$\left.\frac{\frac{\partial R_{SR}(t,k)}{\partial k}}{R_{SR}(t,k)}\right|_{k=k_0} = \frac{\frac{\partial}{\partial R}\left\{R(t)\left[2 - R(t)^{\frac{N}{k}}\right]^k\right\}}{R(t)\left[2 - R(t)^{\frac{N}{k}}\right]} = \tag{1.52}$$

Cu notația: $n = R(t)^{\frac{N}{k}}$, atunci când *1<k<N => 0<n<1* şi valoarea căutată a lui k_0 va fi dată de soluția ecuației:

$$\frac{\frac{\partial R_{SR}(t,k)}{\partial k}}{R_{SR}(t,k)} = \frac{n \cdot \ln n}{2 - n} + \ln[R(t)(2 - n)] = 0 \tag{1.53}$$

Notând suplimentar:

$$y = \frac{n \cdot \ln n}{2 - n} \tag{1.54}$$

$$z = -\ln[R(t)(2 - n)] \tag{1.55}$$

şi reprezentând pe acelaşi grafic, funcțiile y şi z (Fig. 1.27) se constată că cele două grafice se intersectează într-un singur punct (deci există o valoare unica pentru k_0).

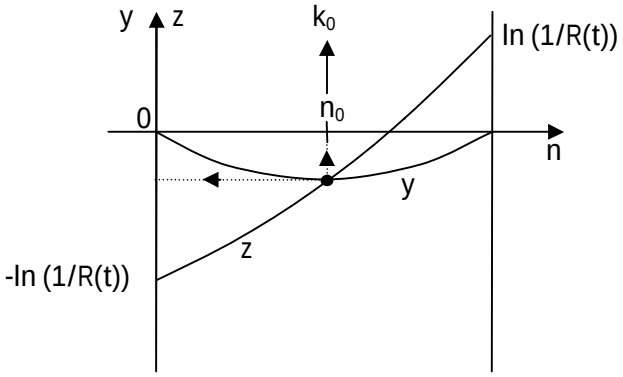


Fig. 1.27 Determinarea grafică a numărului optim de partiționare

Pornind de la această dezvoltare particulară, se poate stabili o metodă generală de calcul al valorii optime pentru numărul de partiționări al unui sistem concret, în vederea implementării redundanței protective, care va cuprinde următoarele etape:

- se calculează derivata funcției de fiabilitate a sistemului redundant în raport cu *k*, numărul de module funcționale cărora li se implementează redundanța;
- se alege valoarea întreagă cuprinsă între două valori consecutive ale lui *k*, pentru care funcția $\frac{\partial R_{SR(t,k)}}{\partial k}$ are semne contrare, care desigur corespunde punctului de maxim al funcției de fiabilitate;
- valoarea optimală a lui $k=k_0$ este dependentă de funcția de fiabilitate a sistemului căruia i se implementează redundanța protectivă la defecțiuni şi implicit de timpul necesar îndeplinirii misiunii.

1.7 CONCLUZII ŞI CONTRIBUȚII

Capitolul 1 al tezei de doctorat intitulat "Tehnici de implementare a toleranței la defecțiuni în sisteme digitale complexe" reprezintă o analiză sintetică a stadiului actual al cercetărilor în domeniul sistemelor autonome dinamice de mare fiabilitate, mentenabilitate şi disponibilitate.

Pe baza studiului efectuat se poate concluziona că implementarea structurilor redundante în vederea asigurării toleranței la defecțiuni, reprezintă cea mai modernă metodă de creştere a fiabilității echipamentelor electronice digitale complexe. În cadrul acestui capitol au fost tratate următoarele probleme principale:

- Definirea şi modelarea conceptului de toleranță la defecțiuni, modelele de defecțiuni cele mai importante utilizate la testarea echipamentelor electronice, precum şi algoritmii actuali utilizați la detecția şi localizarea defectelor sau pentru mascarea defectelor şi reconfigurarea sistemelor electronice digitale.
- Prezentarea structurilor redundante protective pentru implementarea toleranței la defecțiuni: statice prin procedee de multiplicare, digitale cu logică majoritară sau

cvadruplă, prin aplicarea codurilor detectoare şi corectoare de erori, dinamice şi hibride. Se poate constata că structurile redundante protective prin codare şi dinamice oferă cele mai bune rezultate pentru echipamentele digitale tolerante la defecțiuni.

- Analiza principalelor probleme de sincronizare care apar la interconectarea sistemelor digitale tolerante la defecțiuni, cunoscut fiind că diafonia, cuplajele parazite, reflexiile ca şi întârzierile produse pe magistralele de date şi adrese produc numeroase erori şi hazarduri în funcționarea echipamentelor electronice digitale. În finalul Capitolului s-au prezentat selectiv cele mai importante criterii de comparare a performanțelor sistemelor digitale complexe tolerante la defecțiuni;
- Considerăm că la elaborarea ultimului subcapitol am adus cele mai importante contribuții teoretice la implementarea optimală a structurilor redundante în echipamentele digitale tolerante la defecțiuni. S-au analizat principalele strategii actuale de implementare a redundanței şi s-a propus o metodă de stabilire a nivelului optimal de partiționare a unui sistem, pentru aplicarea redundanței protective la defecțiuni.

Principalele contribuții originale aduse de autor la elaborarea acestui capitol pot fi rezumate astfel:

- Analiza şi prezentarea în sinteză a stadiului actual al cercetărilor din domeniul sistemelor digitale autonome dinamice, tolerante la defecțiuni, în scopul identificării direcțiilor viitoare de cercetare.
- Prezentarea în formă compactă şi modelarea conceptului de toleranță la defecțiuni în echipamentele electronice digitale cu structură redundantă protectivă la defecțiuni.
- Analiza şi sistematizarea celor mai importante modele de defecțiuni cunoscute în literatura de specialitate pentru testarea sistemelor electronice digitale.
- Elaborarea unui model matematic pentru analiza toleranței la defecțiuni şi stabilirea diagnozabilității echipamentelor electronice de mare utilitate practică.
- Analiza comparativă a algoritmilor utilizați curent la detecția şi diagnosticarea defectelor în echipamentele electronice digitale.
- Dezvoltarea algoritmilor de reconfigurare a sistemelor şi de mascare a defectelor în echipamentele numerice, prevăzute cu structuri redundante protective de tip static sau dinamic.
- Prezentarea în sinteză a schemelor de implementare a redundanței protective la nivel hardware, ca metodă eficientă de realizare a toleranței la defectări în sisteme electronice digitale.
- S-a demonstrat ca probabilitatea de buna funcționare a sistemelor electronice cu structură redundantă globală este mai redusă decât în cazul sistemelor cu structură redundantă individuală.
- Analiza structurilor redundante protective statice de tip individual sau global, rezultate prin procedee de multiplicare, a permis partiționarea optimală a schemei de bază în unități independente ce pot fi reparate individual.

- Evaluarea întârzierilor în propagarea informației prin modulele sistemului redundant şi limitările performanțelor dinamice pe care le presupun.
- Se recomandă utilizarea structurilor protective statice cu logică cvadruplă care maschează corespunzător defectele la ieşirea sistemului monitorizat, dar prețul de cost creşte uneori nejustificat.
- Bazată pe existența unei varietăți bogate de coduri detectoare şi corectoare de erori, se propune o soluție modernă şi elegantă de realizare a sistemelor automate tolerante la defecțiuni.
- Analiza comparativă efectuată, dovedeşte că structurile redundante protective dinamice şi hibride reprezintă cele mai potrivite soluții de realizare a toleranței la defecțiuni în echipamentele electronice de mare fiabilitate.
- Se propune o clasificare a sistemelor autonome în structuri redundante de tip static, dinamic sau distribuit, ca instrument eficace de elaborare optimală a topologiei magistralelor de date sau adrese tolerante la defecțiuni.
- Analiza detaliată a structurii redundantă dinamic, aplicată magistralelor de date a dovedit că reprezintă cea mai comodă metodă de implementare a toleranței la defecțiuni, comparativ cu alte metode cunoscute, care degradează parametrii dinamici sau limitează capacitatea de transmitere a informației.
- Prezentarea şi analiza defectelor de sincronizare a sistemelor digitale a condus la concluzia că ele limitează aria de aplicabilitate a metodelor clasice de implementare a toleranței la defecțiuni.
- Sistematizarea principalelor criterii de comparare a performanțelor de fiabilitate şi definirea unor criterii noi (indicele de cost şi de eficiență) permite stabilirea celor mai potrivite metode de implementare a toleranței la defecțiuni pentru o aplicație practică bine definită.
- Am elaborat o metodă analitică pentru stabilirea nivelului optim de partiționări a unui sistem digital în vederea aplicării redundanței protective la defecțiuni.

Putem concluziona că pentru a facilita alegerea strategiei optimale de aplicare a redundanței protective la defecțiuni în sisteme digitale, este necesar să se determine şi să se analizeze variația funcției de fiabilitate, raportată la costul solicitat de aplicarea redundanței protective la defecțiuni, în echipamente electronice digitale complexe.

2. ARHITECTURA SISTEMELOR AUTONOME DE CALCUL RECONFIGURABILE

Heterogenitatea şi complexitatea tot mai ridicată a infrastructurilor informaționale ce încorporează aplicații hardware şi software, servicii şi rețele interconectate, au condus la utilizarea unor sisteme de calcul nesigure şi dificil de gestionat. Ca urmare s-a impus abordarea unor studii noi care să permită automatizarea infrastructurii informatice şi managementul optimal al funcționării ei. O soluție viabilă pentru actualizarea problemelor prezentate anterior, o reprezintă un domeniu de cercetare actual şi de perspectivă *„Sisteme autonome de calcul reconfigurabile*", abordat de cele mai importante firme producătoare din domeniu.

Ele dezvoltă şi soluționează cele mai stringente constrângeri de creştere a fiabilității precum: autoconfigurarea, autoprotecția, autorepararea şi autooptimizarea. Cercetările efectuate asupra sistemelor inteligente din ultimii ani: [BC-02], [BS-07b], [BS-07], [CH-05], au permis elaborarea de arhitecturi robuste pentru sistemele autonome de calcul, care să le asigure un comportament autocontrolat şi inteligent asupra mediului informațional.

2.1 ANALIZA MEDIULUI DE CALCUL AUTONOM

În prezent, structura informatică a unei intreprinderi este foarte complexă astfel încât funcționarea, exploatarea şi mentenanța ei, necesită eforturi materiale şi intelectuale considerabile. Ideea construirii unui sistem inteligent de management al structurii informatice a fost pentru mult timp dezvoltată în mediile academice şi industriale. Un grad mai mare de automatizare în supravegherea şi controlul tuturor elementelor structurii informatice este dat de principiile calculatoarelor autonome: autoconfigurare, autoprotecție, autooptimizare şi autoreparare.

Aceste trăsături îi permit adaptarea la gradul de utilizare al resurselor în timpul funcționării, reprezentate de nevoile actuale ale utilizatorului; prin urmare sistemul de calcul trebuie să se optimizeze transparent şi în mod continuu. Aceste sisteme sunt dotate cu mecanisme de învățare [CL-00] şi sunt capabile să se autoadapteze la condițiile de mediu şi la semnalele aflate în continuă schimbare. Acest concept a avut un impact puternic asupra unor domenii precum sistemele de control, robotica, vehiculele ghidate automat, sistemele de control inteligente, şi au fost adaptate la controlul şi supravegherea sistemelor informatice. Lucrări recente [DG-02c], [ME-02] prezintă calculatoarele autonome ca fiind realizate sub forma unor sisteme de control echipate cu traductoare, controlere şi actuatoari, al căror scop îl reprezintă automatizarea structurii informatice, în timp ce dinamica structurii informaționale este descrisă de sisteme matematice.

Acest capitol dezvoltă ideile prezentate în lucrările [MM-03], [BS-06] şi le transpune într-o arhitectură de referința pentru calculatoarele autonome. Această arhitectură de referință îmbină principiile şi metodologiile specifice sistemelor cu funcționare în timp real, transpunându-le în elemente de proces. Perspectiva temporală a unui calculator autonom, este dată de controlul buclei de reacție a unui instrument al unei aplicații IT. Ieşirile unui sistem autonom, sunt considerate matricile comportamentale utilizate pentru evaluarea răspunsului unui proces de calcul, desfăşurat de un calculator autonom, precum încărcarea unității centrale de procesare, timpul de răspuns şi timpul de încărcare a aplicației. Aceste ieşiri sunt măsurate în timp real de module hardware numite traductoare, în timp ce controlerul care implementează diferite reguli, are rolul să mențină constantă parametrii matricilor. Din punct de vedere al actuatorilor, bucla de control descrisă mai sus, utilizează limbaje care reglează, configurează sau care adaptează softul. În Fig. 2.1 se prezintă o structură a elementelor constructive ale unui sistem autonom şi interacțiunea dintre ele. Această arhitectură este asemanătoare unor sisteme de control automat. Deşi abordarea este generală şi poate fi aplicată mai multor situații, se pot analiza în principal timpii de măsurare din aplicațiile multisarcină. Acestea pot fi elemente de tranzacționare în sistemul bancar, de asigurări sau în sistemele informatice.

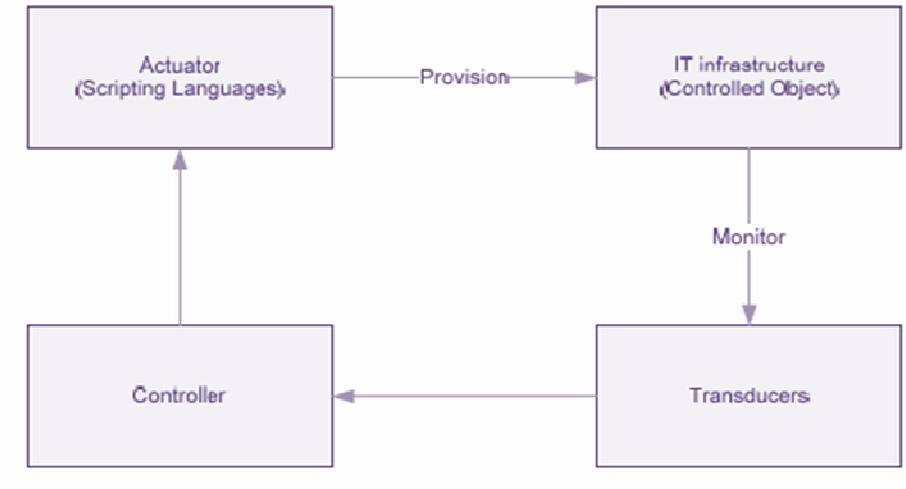


Fig. 2.1 Structura unui mediu de calcul autonom

Această secțiune oferă o privire de ansamblu asupra considerațiilor generale ale arhitecturii sistemelor autonome de calcul. Sunt prezentate în primul rând cerințele unei astfel de arhitecturi şi apoi sunt dezvoltate elementele unui sistem de calcul autonom care alcătuiesc o astfel de arhitectură. Vom aborda şi considerații privind operarea în timp real, necesare în acest context [HC-09].

2.2 ARHITECTURA SISTEMELOR AUTONOME

Cercetările actuale privind elaborarea arhitecturii optimale a unui sistem de calcul autonom pentru o aplicație specifică, pornesc de la elaborarea cerințelor pe baza cărora se construiesc elementele constituente.

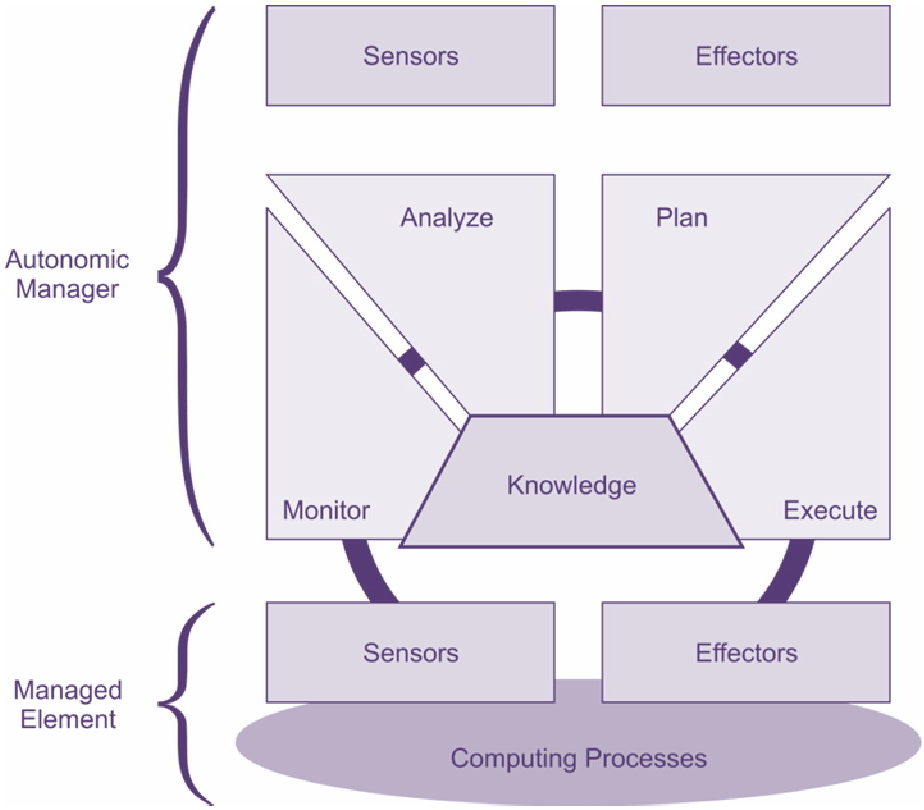


Fig. 2.1: O arhitectura generica pentru sisteme de calcul autonome

Fig. 2.1 prezintă structura de interconectare a unui sistem de conducere autonomă având o singură resursa (denumita un element de conducere). În această arhitectura, resursa poate fi un singur sistem computerizat, o aplicație software găzduită pe un sistem computerizat, sau o colecție de mai multe sisteme conectate logic(un sistem de distribuție computerizat). Senzorii asigură achiziția datelor de măsurare de la resurse şi efectorii, asigură implementarea mijloacelor pentru modificarea timpului de comportare a resurselor.

Conducerile autonome citesc datele senzorului şi manipulează efectorii să facă resursele mai autoconduse. Conducerea autonomă conține componente pentru monitorizare, analiză, planificare şi execuție. Un element comun îl reprezintă este cunoaşterea mediului computerizat, nivelul de servicii agreate şi alte probleme legate de acestea. Componenta monitorizată, filtrează şi corelează datele senzorului. Analiza comportării şi reacția sistemului rafinează seturile de date pentru a face previziuni şi determinări estimative. Componenta de execuție, controlează fluxul de fabricație şi asigură coordonarea, în cazul în care sunt fluxuri de date multiple şi concurente.

Esența arhitecturii autonomă prezentată, asigură o schemă de bază pentru dezvoltarea buclelor de control şi răspuns în sistemele autoconduse. Pe baza acestei scheme este fezabil să utilizăm teoria de control, pentru a asigura ghidarea la structura şi cerințele conducerii autonome multiple.

2.2.1 Cerințe de bază

Ideea de bază a arhitecturii constă în descompunerea unui sistem de calcul autonom în elemente de bază, precum: senzori, actuatori, elemente decizionale etc, pentru a le putea dota cu funcționalitățile şi interfețele standard. Aceste elemente, sunt apoi asamblate folosind diferite scheme specifice sistemelor cu funcționare în timp real. Deşi scopul tezei îl constituie autooptimizarea sistemelor de calcul autonom, elementele prezentate sunt comune şi altor sisteme autonome: autoconfigurarea, autorepararea sau autoprotecția. Avantajul unei astfel de descompuneri este imediat. Elementele de bază pot fi utilizate în proiectarea şi implementarea altor sisteme autonome. Un exemplu ar fi: senzorul utilizat pentru autoconfiguare, care transferă în mod dinamic sarcina sistemului de calcul autonom într-una sau mai multe variabile software, care pot fi folosite în autoreparare, autooptimizare sau în autoprotecție.

Un alt concept folosit în proiectarea şi implementarea arhitecturii sistemelor autonome de calcul îl constituie distributivitatea. În cadrul acestei arhitecturi elementele componete folosite pentru a implementa un sistem de calcul autonom, se pot regăsi în orice nod al sistemului distribuit.

De obicei, entitățile software care îndeplinesc funcțiile sistemelor de calcul autonom aparțin unei platforme complexe care constituie suportul unui proces de calcul masiv şi care devine dificil de gestionat, dacă funcționează pe baza unui sistem centralizat. În loc să existe un bloc masiv, care procesesază informația provenită de la senzori si terminând cu cea de la actuatoare, informația circulă printr-un flux de componente dispuse în noduri separate. Prin urmare, caracteristici ale sistemelor în timp real precum memoria distribuită, capacitatea de observare, distributivitatea, devin foarte utile.

Mai mult dect atât, arhitectura propusă trebuie să fie capabilă să se modifice singură în timpul funcționării, deoarece s-ar putea adăuga, înlocui sau muta, elemente in mod dinamic, după cum ele sunt solicitate de sistem. Dacă sistemul automat de gestiune, trece dintr-o stare în alta, sistemul nu poate numai să reconfigureze infrastructura, ci se poate configura singur pentru o mai bună gestionare a infrastructurii. De exemplu, sistemul poate utiliza diferite controlere de-a lungul utilizării sale, începând cu cele elementare, până la dele mai complexe sau adaptive, pe masură ce sistemul se adaptează şi se configurează la mediul său.

Un alt exemplu de autoadaptare îl reprezintă crearea dinamică şi izolarea traductoarelor, pe masură ce sunt adăugate/îndepărtate componente. Unul dintre avantajele autoadaptării îl constituie accesul sistemelor de gestiune autonomă la noi componente, atunci când are nevoie de ele, şi să le îndepărteze când nu mai sunt necesare. Prin urmare, nu numai infrastructura gestionată, ci şi sistemul de gestiune autonom poate funcționa la solicitările aplicației.

2.2.2 Componentele sistemului de control

Generalizând rezultatele prezentate în lucrările [MM-03] şi [BS-06], se propune un set de şapte elemente de bază pentru construcția unui sistem de calcul autonom, fiecare dintre ele având o funcționalitate clară, bine definită şi distinctă:

- **Componenta-senzor** – este responsabilă de sesizarea şi achiziția datelor de la entitatea sau entitățile gestionate şi care descriu in mod cantitativ starea sistemelor gestionate. O astfel de componentă cuprinde mai mulți senzori, daca se urmăreşte supravegherea unor entități diferite.
- **Componenta-filtru** – filtrează date provenite de la senzori prin eliminarea zgomotului sau pentru efectuarea modificărilor solicitate de controlerul care le prelucreaza. Componenta-filtru implementează un lanț de filtre care permite prelucrarea datelor printr-o serie de filtre consecutive.
- **Componenta-coordonator** – organizează procesul de predicție. Ea recepționează datele de la filtre, utilizează blocul de modelare, estimatorul şi blocul decizional, pentru a lua o decizie predictivă şi trimite această decizie actuatorilor. Odată cunoscută decizia, se adaugă sau se înlătură senzori.
- **Blocul modelator** – reține modelul entității sau entităților generate. Este utilizat de estimator şi de blocul decizional. El poate actualiza modelul dat pe baza unor estimări apriorice sau a unor modificări survenite asupra infrastructurii modelate.
- **Estimatorul** – realizează estimarile viitoare ale stării sistemului modelat pe baza stării actuale, sau dacă este necesar, utilizează informații referitoare la stările anterioare.
- **Blocul decizional** – formulează decizii predictive bazate pe datele estimate.
- **Blocul actuator** – activează starea actuatorilor pe baza semnalelor furnizate de blocul decizional. El cuprinde mai mulți actuatori.

În timp ce un sistem autonom poate fi construit pe baza a şapte componente, arhitectura nu impune ca toate sistemele autonome să utilizeze toate componentele predefinite. Proiectanții sistemelor autonome hotărăsc ce blocuri trebuie utilizate.

Logica buclei de control a sistemului poate fi privită ca un subsistem format din coordonator, estimator şi blocul decizional. Subsistemul GUI, de care nu se ocupă acestă lucrare, ar fi prezentat utilizatorului modelul infrastructurii. Infrastructura comunicațională care poate fi refolosită între componente, reprezintă în esență un subsistem care se ocupă de schimbul de mesaje dintre componente.

Prin urmare se oferă un nivel de abstractizare pentru comunicația între componente şi se reduce numarul legăturilor între elemente sau subsistemele care comunica între ele. În Fig. 2.3 se prezintă subsistemele componente ale sistemelor de calcul autonom.

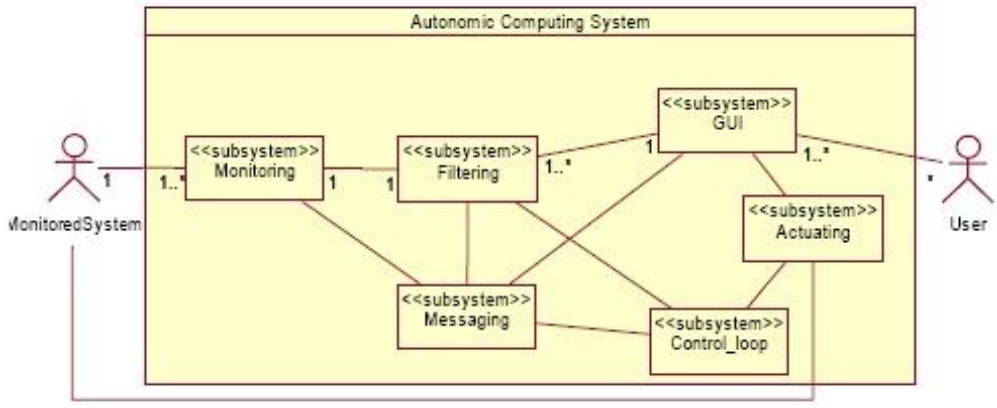


Fig. 2.3 Structura subsistemelor

În Fig. 2.4 este analizată structura pe componente(module) a sistemului şi modul în care acestea comunică între ele. Şablonul fiecărei componente prezintă amprenta subsistemului de care aparține.

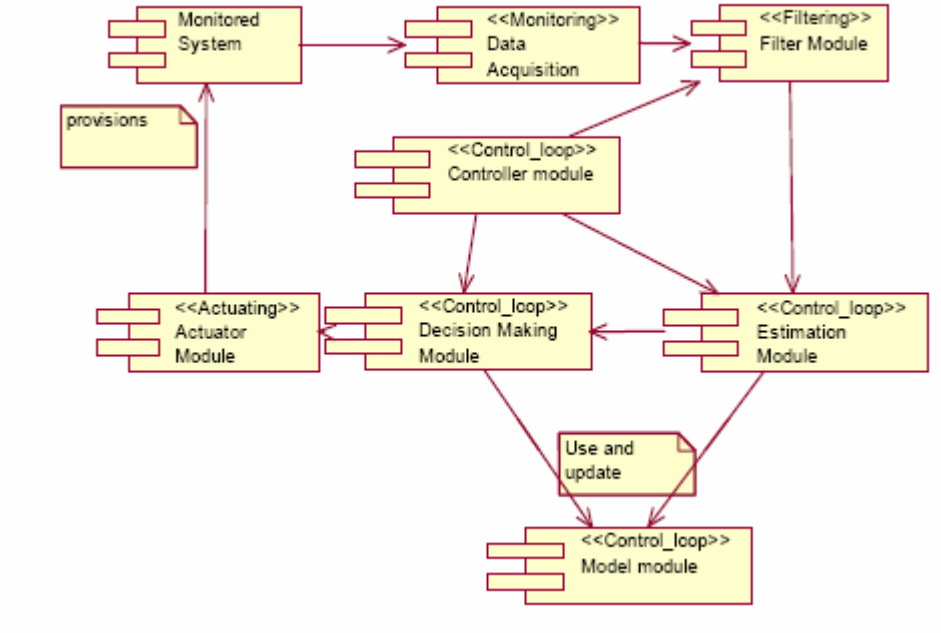


Fig. 2.4 Structura pe componente a subsistemelor

2.2.3 Soluții de implementare în timp real

Arhitectura propusă în această teză se bazează pe conceptul unei modelări şi optimizări a funcției de bază a componentelor din sistem. Rolul acestui model software este de a oferi o soluție alternativă pentru modelul componentelor interschimbabile. Modelul propune o serie de soluții care ilustrează cerințele sistemului: izolarea componetelor defecte, facilitarea reutilizării componentelor şi diversificarea funcțiilor specifice fiecărei componente. În arhitectura descrisă, pe lângă blocurile de bază care compun un sistem autonom(fiecare dintre ele având o funcționalitate clară si bine definită), sunt create şi componentele separate independente. Comunicarea cu alte componente este realizată printr-un set de obiecte predefinite. Soluția propusă a condus la crearea unor componente independente şi înlocuibile, care sunt prin urmare capabile să funcționeze pe noduri separate ale sistemelor de calcul autonom.

Modelul bazat pe simularea componentelor, este prezentat în Fig. 2.5.

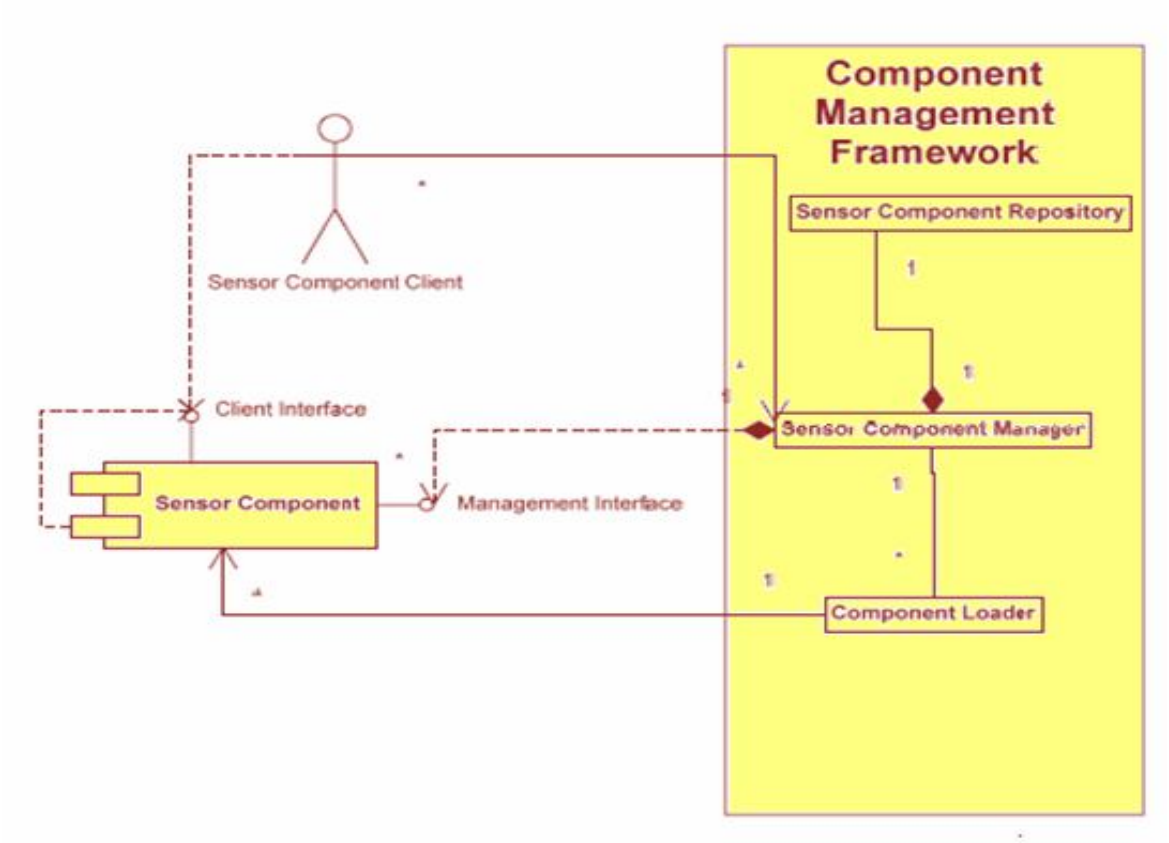


Fig. 2.5 Senzorii şi gestiunea lor

În această schemă componenta-senzor defineşte două interfețe, una pentru gestiune, care este utilizată în timpul creării componentei şi una pentru client, care reperezintă API si este utilizabilă de orice client extern, precum şi de alte module. În acelaşi timp, o fereastră de componente este utilizată pentru creerea şi gestiunea componentelor.

Această fereastră este compusă din trei entități:

- **Gestiunea componentelor**, care este responsabilă de gestionarea componentelor în timpul funcționării, prin încărcarea, înlăturarea sau modificarea componentelor. Este folosită interfața de gestiune oferită de amprenta componentei.
- **Managementul componentelor,** încarcă componenta ca urmare a unei cereri formulată în acest sens.
- **Depozitul de componente** memorează momentele la care au fost încărcate componentele. O cerere pentru o anumită componentă trece prin acestă entitate care încearcă să localizeze componeta şi funizează o soluție de accesare a ei.

Modulul client reprezintă orice client al blocului senzor care foloseşte interfața client a senzorului pentru a recupera informațiile date de senzor. Mai mult decât atât, blocul senzor-client comunică cu blocul senzor-gestiune pentru a cere crearea sau izolarea unui anumit senzor sau pentru a găsi o componentă. În acelaşi timp, un senzor poate utiliza interfața cu clientul pentru dialogul cu alți senzori, dacă aplicația solicită ca senzorii să comunice între ei. În acest mod, un senzor poate deveni client al altui senzor.

Blocul de gestiune a senzorilor supervizează durata de viață a acestei componente, şi orice cerere către o componentă trebuie să treacă prin blocul de gestiune, care este responsabil de localizarea componentei în blocul de depozitare. Din moment ce componentele nu pot fi accesate fără blocul de gestiune, se poate spune despre componente că sunt monitorizate de blocul de gestiune.

Arhitectura prezentată utilizează alte modele software în timp real care se ocupă de soluționarea problemelor de concurență şi fiabilitate care pot apare intr-un sistem complex. Deoarece componentele pot funcționa independent în noduri separate, sistemul trebuie să fie capabil să sincronizeze mai multe componente şi să se asigure că este respectată ordinea corectă de procesare. De asemenea, sistemul trebuie să analizeze integritatea datelor care sunt vehiculate între componente şi să protejeze sistemul de defecțiuni, atunci când unele componente sunt înlocuite sau adăugate.

Pentru a putea înlocui unele componente, este necesar ca starea curentă a componentei ce va fi izolată, să fie salvată şi apoi starea sa fie încărcată de noua componentă. Orice componentă trebuie să aibă starea conservată. Un exemplu de astfel de componentă este un estimator care stochează valorile anterior estimate pentru a aproxima mai bine starea viitoare a sistemului. Când acest estimator urmează a fi înlocuit, valorile anterioare estimate sunt transferate componentei nou creată, pentru a putea conserva starea anterioară. Totuşi există componente a căror stare nu trebuie conservată. De exemplu, un actuator nu are specificată nici o stare stabilă, atât timp cât nu trebuie să ia măsuri de previzionare. Pentru a asigura sincronizarea comunicației dintre mai multe componente, arhitectura implementează modele de concurență în timp real. In arhitectura prezentată, schimbul asincron de mesaje între componente se face pe baza unui şir, iar schimbul sincron de mesaje se face pe baza unui arbitraj.

În cazul şirului, mesajele sunt trecute prin nişte fluxuri de date în care se înmagazinează mesajele care urmează să fie transmise sau recepționate, până când emițătorul sau receptorul solicită datele curente. Integritatea datelor este garantată prin folosirea sferelor de excluziune mutuală, care previn accesările multiple şi simultane ale şirului. Aplicarea principiului de comunicație descris anterior, este prezentat schematic în Fig. 4.6.

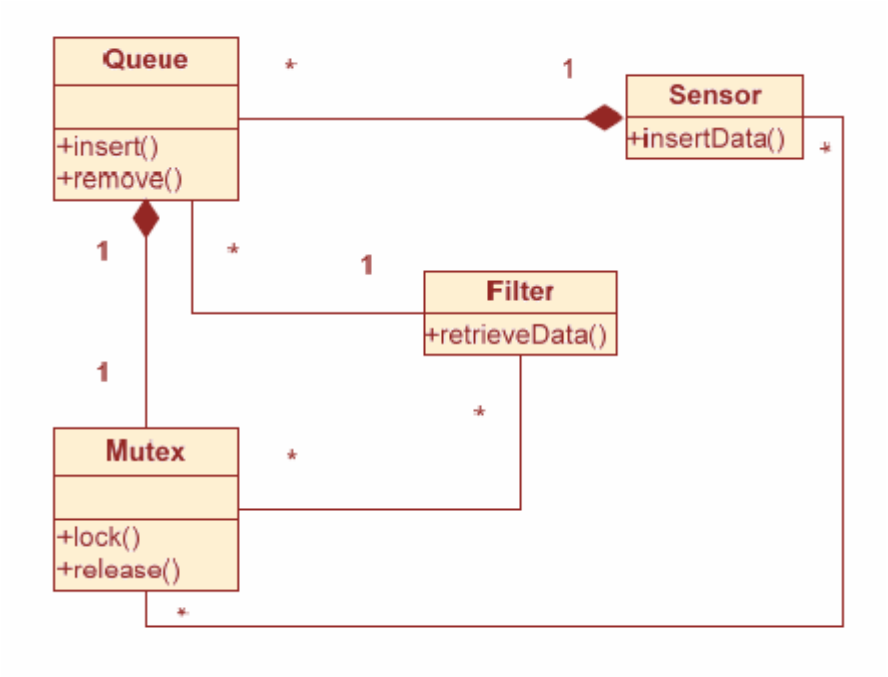


Fig. 2.6 Comunicația senzor-filtru

În schema de mai sus, două sau mai multe fire prin care comunică pachete de mesaje împart un fir(party line) pentru fiecare tip de mesaj care solicită emisia sau recepția. De exemplu, în cazul unui server web, doi senzori diferiți aflați pe două noduri distincte, vor accesa serverele aferente pentru informații asupra utilizării. Aceste informații trebuie apoi trecute prin filtre, înainte de a fi trimise controler-ului. Deoarece atât filtrul cât şi senzorul necesită acces la şir, trebuie să utilizeze amândoi aceeaşi sferă de integritate, pentru asigurarea integrității datelor din şir. Sfera trebuie să fie capabilă să elibereze un proces sau un fir care aşteaptă să fie blocat în şir. Prin urmare, este capabil să semnalizeze fie filtrului, fie senzorului momentul în care şirul a devenit disponibil. Deoarece şirul este folosit pentru transmiterea datelor de la senzor la filtru [BS-07a], este normal să se elibereze şirul când senzorul care îl utiliza este izolat.

După cum s-a arătat mai sus, o sferă de proximitate trebuie folosită pentru acest tip de comunicație în vederea asigurarii integritatii datelor vehiculate între componente. Dacă nu s-ar utiliza o astfel de sferă, atât filtrul cât şi senzorul ar putea accesa şirul în acelaşi moment de timp. Într-o astfel de situație, dacă filtrul ar accesa şirul în timp ce senzorul actualizează datele cu noile informații, filtrul ar recepționa informații parțiale care nu ar reprezenta starea actuală a sistemului gestionat. În unele situații, filtrul ar interpreta datele lipsă ca zero (nule), fapt ce ar introduce erori suplimentare în bucla de

control. Mai mult, astfel de informații incomplete ar duce la erori la prelucrare sau la apariția unei excepții date de filtru, dacă nu pot recupera informațiile necesare procesării ulterioare a datelor. În această situație, folosind o sferă de proximitate, dacă filtrul încearcă să citească datele în timp ce senzorul adaugă noi informații în şir, filtrul ar fi blocat până când senzorul ar termina de adăugat informațiile în şir şi o decizie nu se poate lua pe baza informațiilor curente(deoarece datele ar fi incomplete).

Modelul prezentat, pe baza unui arbitraj [GG-05], se ocupă de cazurile în care un mesaj sincron este necesar pentru comunicația dintre componente. Şirurile de mesaje pot fi folosite doar pentru mesajele asincrone pe masură ce informațiile sunt adăugate în şir şi citite prin buffere la momentul potrivit. Această situație determină ca datele adăugate în şir să fie procesate de receptor la următoarea execuție. În exemplul anterior, nu era important ca schimbul de mesaje între senzori şi filtru să fie sincron atâta timp cât el are loc periodic şi perioada utilizată este suficient de mică pentru secvența de control. Există însă situații când în cadrul sistemelor de calcul autonom sunt necesare mesajele sincrone. Un astfel de exemplu este comunicația dintre un coordonator şi un model. În aplicația de test, coordonatorul foloseşte o abordare iterativă pentru a estima informațiile şi pentru a actualiza modelul. La fiecare iterație, coordonatorul estimează informațiile, actualizează modelul şi verifică dacă valorile estimate sunt apropiate. Mai mult decât atât, estimatorul utilizează modelul pentru a calcula estimatele. Comunicația între componente în acest caz trebuie să fie sincronă, deoarece coordonatorul trebuie să aştepte întoarcerea de la estimator a datelor pentru model inainte de a-şi continua funcționarea. Cea mai simplă soluție este de a găsi o metodă de apelare a componentei potrivite. Aceasta ar conduce la situații de excludere mutuală, astfel că modelul de apelare impune utilizarea unei sfere de proximitate pentru a preveni accesarea simultană a datelor curente. În Fig. 2.7 este prezentat modul de aplicare al acestui model în arhitectura calculatoarelor autonome.

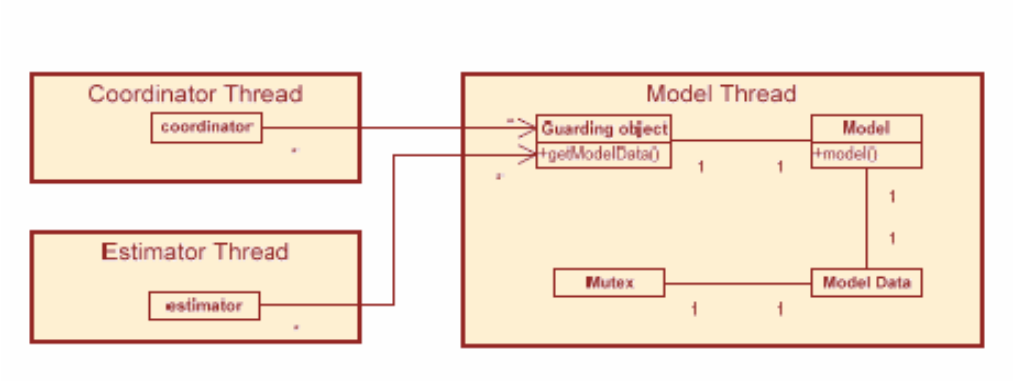


Fig. 2.7 Comunicația dintre estimator, coordonator şi model

Se identifică trei componente separate: un coordonator, un model şi un estimator. Clientul încerac să acceseze o resursă aflată în server. In exemplul dat, coordonatorul şi estimatorul încearcă să acceseze simultan datele conținute în modelul componentelor. Apelarea se face prin server şi se realizează cu ajutorul unui obiect de gardă. Acesta are rolul de a îmbina toate metodele de apelare care utilizează aceleaşi resurse partajate, astfel încât acestea să fie protejate în situația în care sunt apelate simultan de alți utilizatori. La fel ca in cazul utilizării şirururilor, sfera de proximitate reacționează ca un semafor, protejând resursele partajate de apelări multiple.

Acest model este utilizat curent deoarece oferă o posibilitate de optimizare a mesajelor sincrone (utilizarea şirurilor nu oferă această facilitate deoarece utilizează mesajele asincrone). Spre deosebire de modelul bazat pe întreruperi care presuspune ca funcția de răspuns să fie rapidă, acest model permite întâzieri mai mari ale timpilor de calcul şi prin aceasta evită erorile de prelucrare.

Cea de a doua problemă pe care modelul de sistem în timp real ar trebui să o rezolve în arhitectura propusă, este cea a fidelității, caracteristică de bază a auto-reparării sistemelor autonome. Cerințele de priectare ale sistemului prevăd adăugarea, înlăturarea şi modificarea dinamică a componentelor, iar arhitectura trebuie să le facă față. De exemplu, componentele sistemului trebuie să işi continue funcționarea chiar dacă o componentă ce comunică direct cu ele este izolată pentru modificări, sau trebuie să fie capabile să facă față tuturor schimbărilor apărute în interfața utilizată pentru comunicația între ele. In acelaşi timp datele vehiculate între componente trebuie să fie protejate de alterări.

Pentru a asigura integritatea datelor vehiculate între componente, se foloseşte modelul canalului protejat. Scopul acestui model este de a adăuga puncte de validare în anumite secțiuni de pe canalul de comunicație. Acest proces începe de la senzori şi se încheie la actuatori. De exemplu, înainte de a comanda de închiderea a unui server, sistemul ar trebui să verifice dacă închiderea server-ului nu ar cauza pierderea datelor sau scoaterea din funcțiune a unor elemente. În Fig. 2.8 este prezentat modul în care verifcarea canalului se aplică pentru sistemele de calcul autonom.

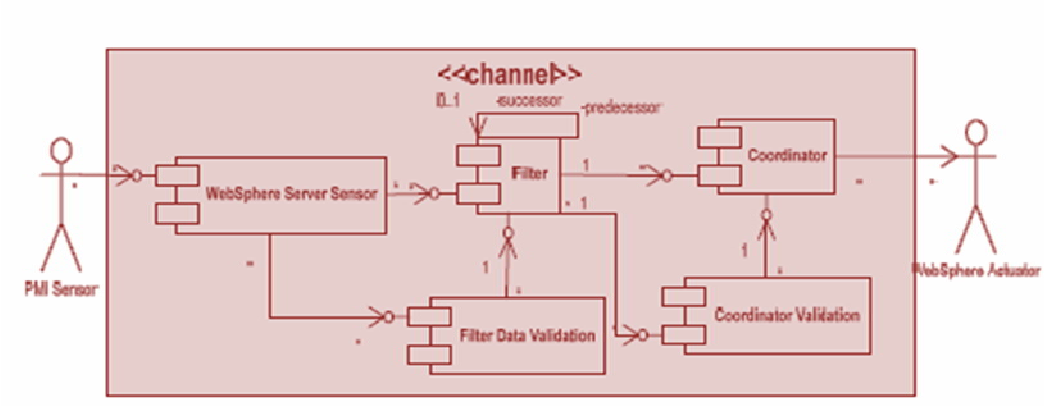


Fig. 2.8 Verificarea canalului

Intrarea senzorului este prelucrată de unitatea de procesare a datelor pentru a extrage datele inițiale. Având extrase aceste informații, ele sunt transmise unității de transformare care elaborează modelul de acționare al actuatorilor. În anumite puncte ale canalul de comunicație, exista unități care verifică corectitudinea datelor şi starea sistemului pentru a evita hazardurile în funcționare. Datele recepționate ajung la unitatea de procesare a semnalelor care indică actuatorilor modul concret în care trebuie să acționeze.

2.2.4 Interoperabilitatea şi comunicația

Infrastructura informațională este una dintre cele mai importante elemente ale sistemului, ea asigurând existența unui flux de date corect prin sistemul de gestiune. În sistemul de calcul autonom descris, comunicația între componente s-a realizat prin intermediul serviciilor web. Fiecare componentă funcționează ca un terminal web care acționează atât ca interfață pentru client, cât şi ca interfață pentru sistemul de gestiune, conform Fig.2.3. Descrierea interfeței s-a făcut în limbajul Web Service Description Language (WSDL) [5]. Prin utilizarea serviciilor web pentru infrastructura de comunicație între componente, s-a asigurat utilizarea de către sistem a unui standard deschis şi acoperirea de către componentele sistemului a mai multor limbaje.

Serviciile web se datorită posibilității de extindere a sistemului adăugând un registru Universal Description, Discovery and Integration (UDDI) care permite interfațarea şi conectarea dinamică între componente. Serviciile web se recomandă şi datorită posibilității de utilizare a serviciilor OASIS Web Service Distributed Management (WSDM) care simplifică gestionarea resurselor în cadrul terminalelor web.

2.3 ARHITECTURA MODELELOR PLATFORMĂ

Dezvoltarea sistemelor de control pentru echipamentele autonome de calcul [CM-05], [DC-03] au permis elaborarea de arhitecturi bazate pe modelul platformelor: modelul platformă independentă (Platform Independent Model – PIM), modelul platformă specifică (Platform Specific Model –PSM) şi în final pe baza lor, generarea codului executabil. PIM descrie arhitectura sistemului fără a detalia platforma. Prin urmare, a fost dezvoltat un model general pentru reprezentarea structurii şi pentru comportamentul buclei de control autonome.

2.3.1 Platforma-model independent (PIM)

Pentru a putea defini un model platformă independent este necesar să se determine domeniile şi clasele care reprezintă spațiul problemei şi al soluțiilor. Partea centrală a modelului este sistemul gestionat care reprezintă mediul ce trebuie automatizat şi care conține informație hardware şi software, cerințele automatizării precum obiectivele la nivel de servicii (Service Level Objectives) şi metodele de predicție. Modelul domeniului este reprezentat de structura buclei de control care este împărțită în: domeniul traductoarelor, al avertizărilor şi al mijloacelor de execuție care conține estimatorul şi blocul decisional, dar şi domeniul efectorului care conține actuatorii, factorii şi mijloacele de execuție. Toate domeniile buclelor de control utilizează un domeniu de comunicații care asigură schimbul de mesaje între diferite componente ale domeniilor (Fig 2.10).

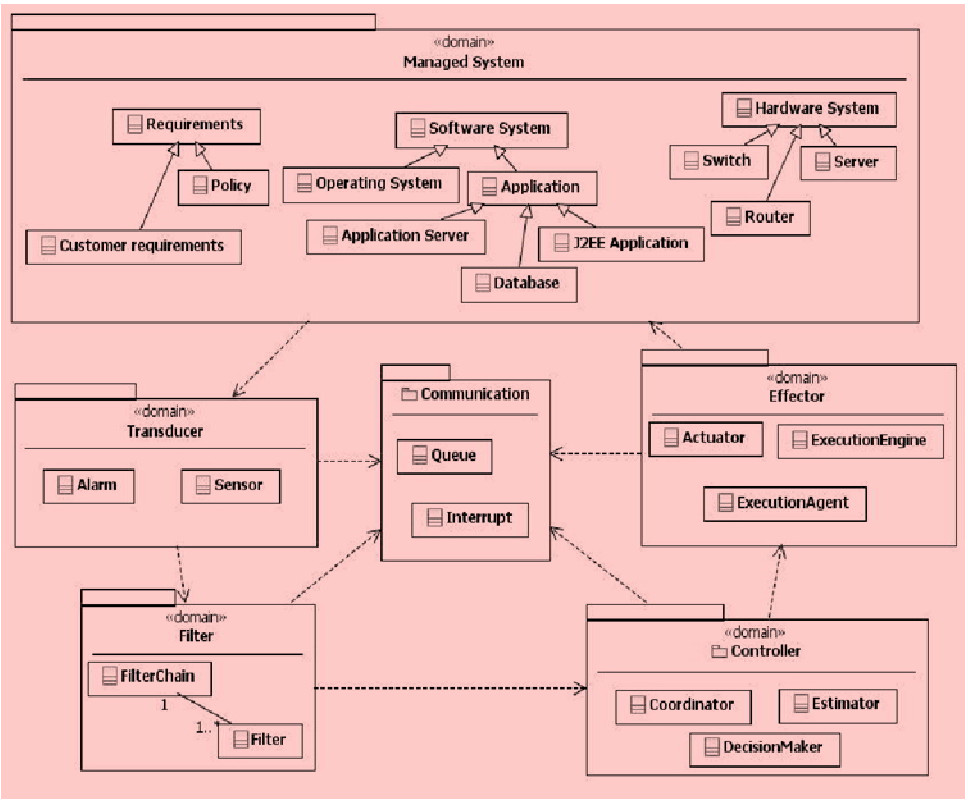


Fig.2.9 Modelul domeniului

Modelul poate fi descompus în componente discrete, fiecare dintre ele având o funcționalitate clară şi distinctă. Sistemul autonom prezentat în Fig. 2.10 se compune din şapte componente: senzor, filtru, coordonator, modelul entităților gestionate, estimatorul, blocul decizional şi actuatorul, care inițiază acționările (pe baza deciziilor predictiveluate lanivelul blocului decizional)

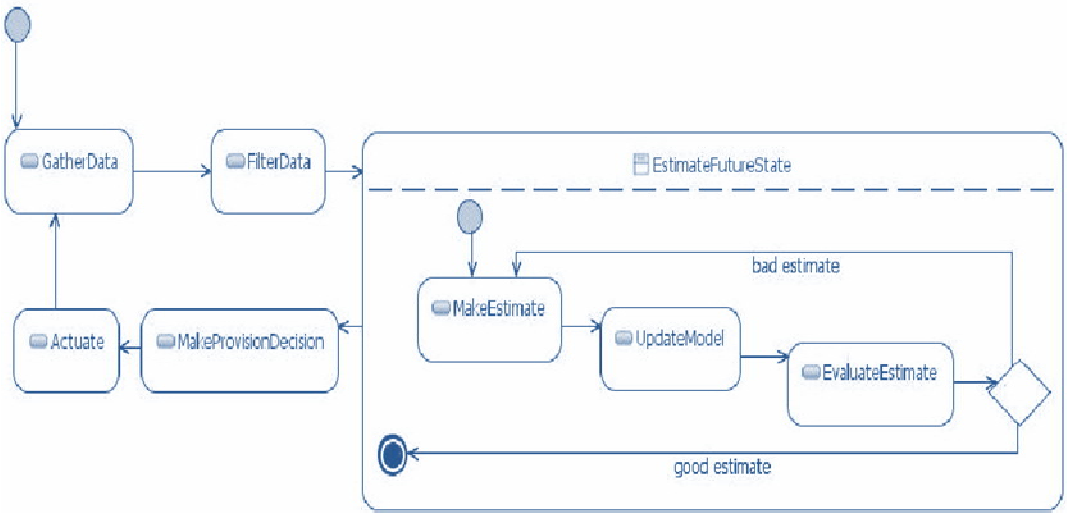


Fig. 2.10 Schema de interconectare a componentelor PIM

Pentru a realiza sistemul autonom reconfigurabil, astfel încât ciclul său de viață trebuie să fie utilizat pentru automatizarea generării codului, iar comportamentul sistemului trebuie să fie reprezentat printr-o arhitectură PIM. Pentru aceasta, Fig. 2.11 prezintă tabelul general al stărilor, iar Fig. 2.12 prezintă diagrama stărilor pentru schimbul de mesaje dintre componente [DC-04b], [DG-02a].

După cum se constată în diagrama stărilor, bucla de control începe prin adunarea informațiilor de la entitatea gestionată. Toate informațiile necesare sunt filtrate, iar pe baza lor poate face un estimat al viitoarelor stări ale sistemului. Pentru a putea estima viitoarele stări, se poate utiliza o abordare iterativă în care starea curentă este estimată, pe baza stării anterioare. Dacă estimatul este acceptat atunci bucla de estimare se închide; altfel încât buclarea continuă până se găseşte un estimat optimal (sau până se consumă un anumit număr de iterații fără a obține un estimat suficient de bun). Odată estimată starea viitoare, pe baza estimării făcute sistemul decide ce măsură trebuie aplicată entității gestionate. Sistemul începe apoi o altă iterație a buclei de control, cu o anumită întârziere.

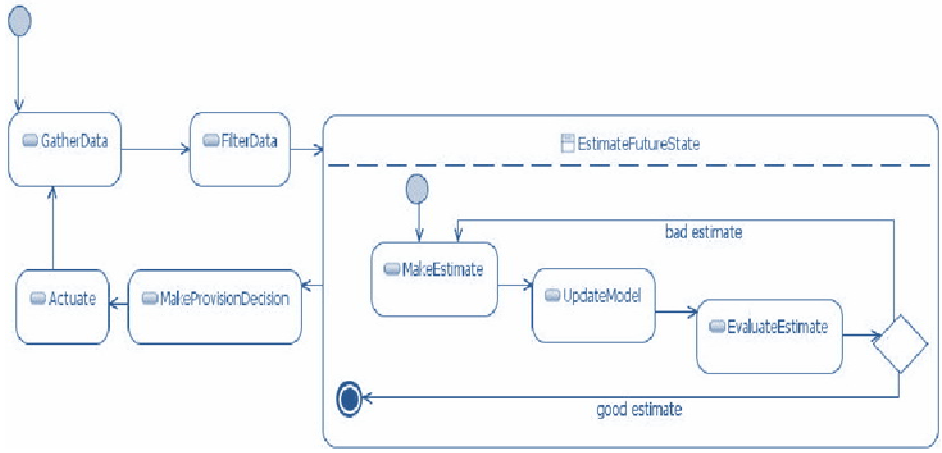


Fig. 2.11 Schema generală de stări a PIM

Diagrama stărilor prezintă secvența de mesaje dintre componente. Odată adunate informațiile provenite de la mediu şi de la sistem, senzorii le trmit filtrelor care le filtrează şi sunt disponibile coordonatorului, care este responsabil de funcționarea logicii buclei de control. El actualizează modelul cu noile informații şi cere estimatorului să determine starea viitoare a sistemului. Estimatorul iterează estimatul până când găseşte un estimat optim şi stabil, returnând apoi acest estimat coordonatorului. În cele din urmă, coordonatorul solicită blocului decizional să transmită actuatorului mecanismul care trebuie aplicat entității gestionate.

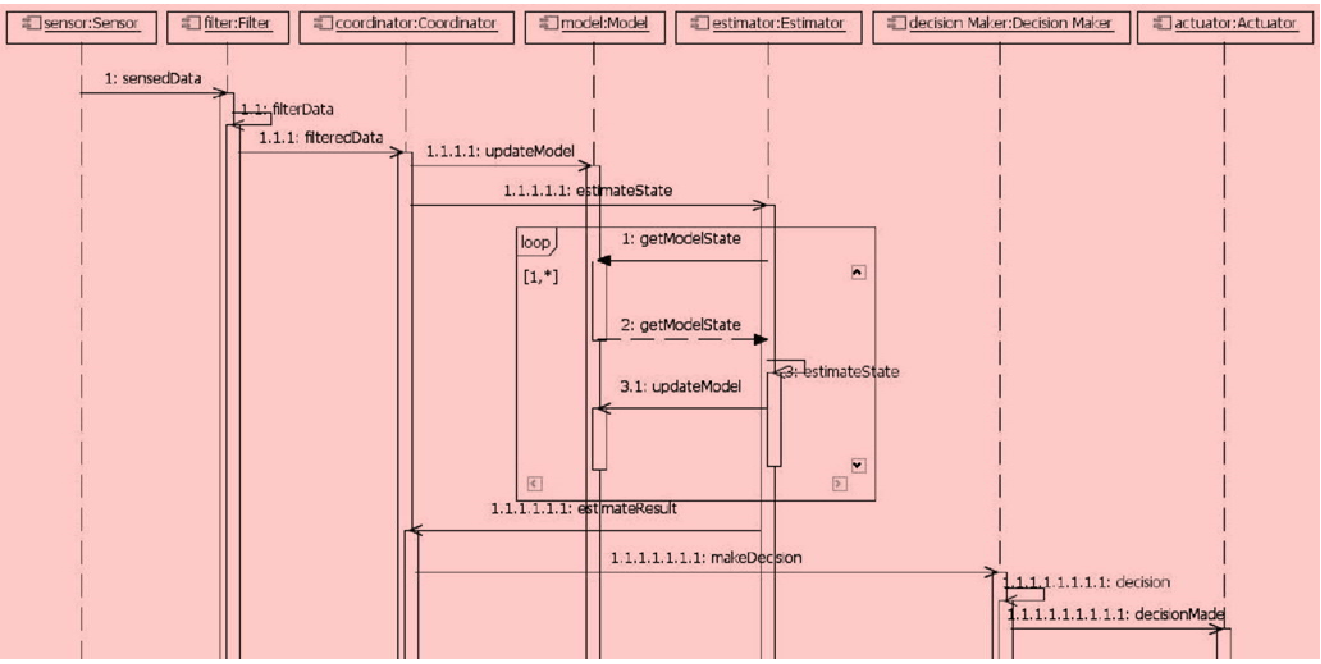


Fig. 2.12 Diagrama stărilor PIM

Fluxul de date poate fi proiectat cel mai bine utilizând modelul arhitecturii canalului (Channel Architecture Pattern) [CB-03] pentru a furniza comunicații fidele.In modelul canalului datele din flux sunt procesate secvențial urmând o serie de etape prestabilite. Pentru sistemele autonome, fluxul de date este divizat într-un număr de canale :

- **Canalul senzor şi filtru**- informațiile sunt măsurate de senzori şi sunt trimise filtrelor care le procesează şi le trimit mai departe coordonatorului
- **Canalul coordonator**- informațiile provenite de la mai multe canale de filtrare sunt procesate de estimator şi de model în mod secvențial
- **Canalul actuator**- informațiile estimate sunt utilizate pentru creerea unei decizii iar datele decizionale sunt folosite pentru execuția unei acțiuni autonome.

Fig. 2.13 prezintă cele trei canale şi interacțiunea dintre ele.

2.3.2 Platforma-model specific (PSM)

Pe baza platformei-model independente descrisă în secțiunea 2.3.1 a fost dezvoltată o platformă-model specifică bazată pe Java Web Service Distributed Management (WSDM) [AT-02].

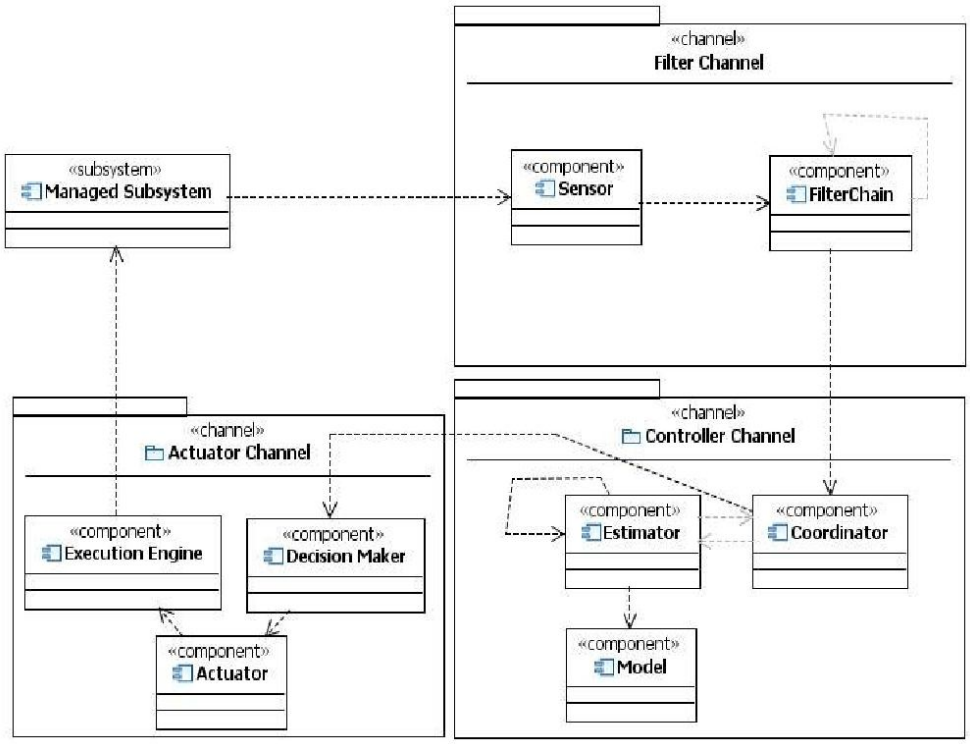


Figura 2.13 Modelul canalului

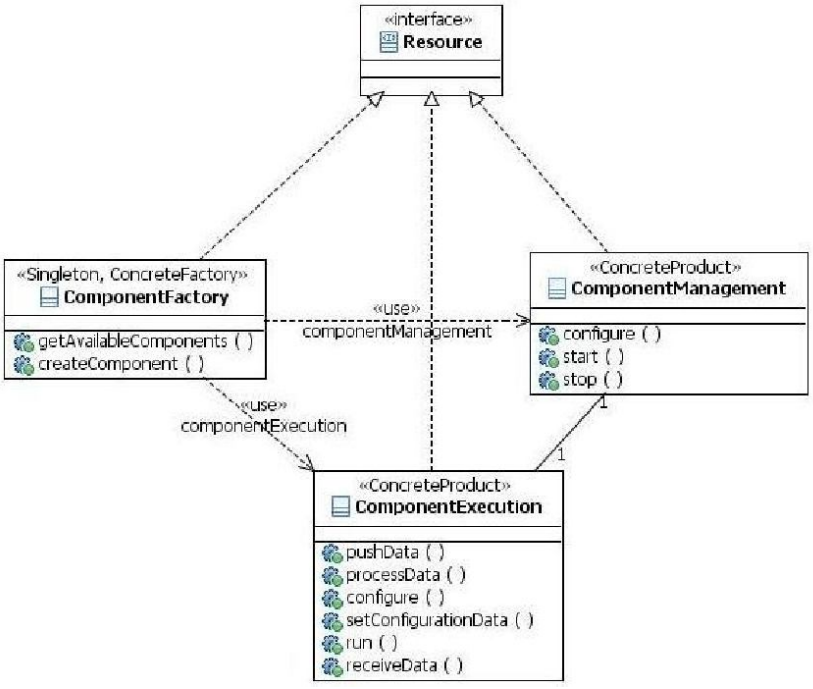


Figura 2.14 Componenta PSM a WSDM

În platforma-model specificată, fiecare componentă se reflectă într-o sursă WSDM. Mai mult decât atât, resursele pot fi distribuite pe mai multe servere J2EE şi comunicația între servere este realizată prin intermediul Web Service Addressing (WS-A) şi prin Simple Object Access Protocol (SOAP).

Spațiul WSDM în care evoluează fiecare componentă este responsabil de asigurarea comunicației între componente, precum şi de functionalitățile suplimentare, precum crearea la cerere prin intermediul unui model sau accesarea componentelor la revenirea în urma remedierii defectelor.

Fiecare componentă este subdivizată în trei resurse:

- **Generatorul de componente**- resursă unică pentru fiecare spațiu WSDM responsabil de creerea de componente care rulează în cadrul lui
- **Gestionarea componentelor**- este resursa pentru fiecare componentă care asigură gestiunea resurselor prin configurare, sau capacitatea sistemului de a reazliza performantele la nivelurile impuse.
- **Rularea componentelor**- resursa care asigură pentru fiecare componentă execuția logicii asociate ei

Fig. 2.13 prezintă cele trei părți care alcătuiesc o componentă autonomă bazată pe WSDM. Toate cele trei clase implementează interfața resursei (Resource Interface) şi care este parte integrantă a specificațiilor WSDM. Generatorul de componente este unica modalitate care permite clientului să solicite componentele care trebuie create de acest generator şi de asemenea îi permite clientului să genereze noi componente. Crearea unei noi componente generează de fapt cele două clase asociate, pentru gestionarea şi rularea componentei. Deşi componenta este creată, ea nu va fi configurată şi nu va fi utilizată până nu va fi primit acceptul de reconfigurare.

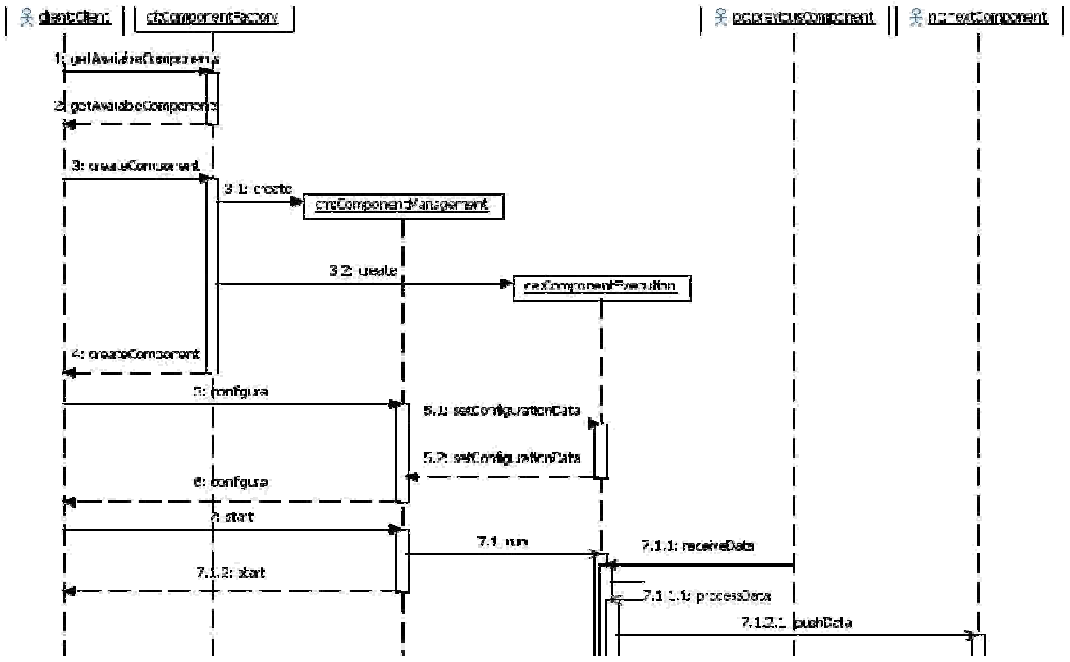


Fig. 2.15 Diagrama stărilor PSM

Partea de gestiune permite clientului să configureze resursa, să pornească şi să oprească rularea resursei. Resursa rulată creează unul sau mai mulți receptori pentru evenimente ale componentelor care sunt în amonte de el în cadrul canalului. Are posibilitatea de procesare a datelor recepționate şi să trimită informațiile către următoarea componenta din canal. Funcția de procesare a informațiilor (ProcessData Function) este un element de bază in funntionalitatea componentei. De exemplu, un filtru o va utiliza pentru transferul datelor printr-un lant de filtre, in timp ce estimatorul o va folosi pentru estimarea stărilor viitoare.

În această concepție fluxul de date este realizat prin intermediul notificarilor dintre componente, iar comenzile sunt transmise prin RPC sau SOAP.

Fig. 2.14 prezintă diagrama secvențială a stărilor pentru ciclul de viață al unei componente. Inițial, un client îi cere generatorului să creeze toate componetele pe care acesta este capabil să le facă. Dacă un client cunoaşte exact tipul componetei dorite, se poate sări această etapă. Următorul pas este crearea componentei. Odată creată componenta, ea trece în starea de stand-by, caz în care acceptă doar un mesaj de configurare. Clasa de gestiune ajustează pe calea de rulare parametrii de configurare necesari. De exemplu, dacă componenta este un senzor, unul dintre parametrii de configurare va fi perioada de eşantionare. Această valoare va fi transmisă clasei de rulare pentru a fi utilizată în timpul execuției. Fig. 2.15 prezintă structura componentelor PSM din WSDM.

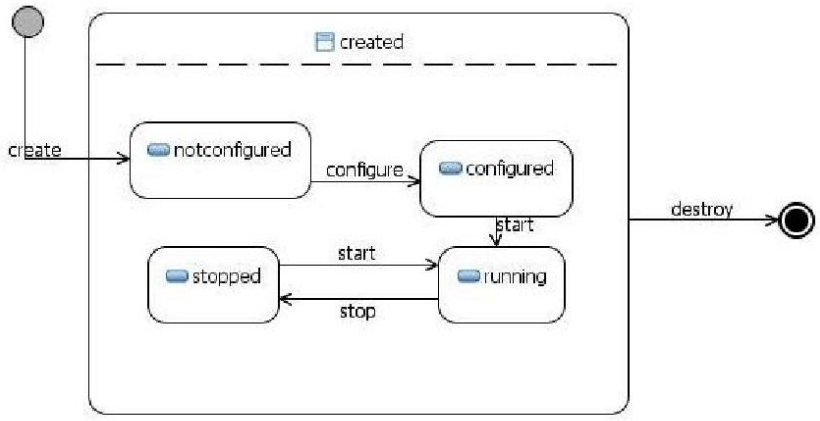


Fig. 2.16 Componenta PSM a WSDM

Odată componenta configurată, ea poate fi accesată sau izolată prin intermediul mesajelor trimise resursei de gestiune. În timpul funcționării, componenta poate receptiona mesaje de la componentele aflate în amonte de ea pe acelaşi canal, poate procesa informațiile receptionate conform resurselor sale interne şi poate trimite informațiile prelucrate următoarei componente din lanț.

În caz de defecțiuni sau incapacitate de preluare a mesajelor, componenta reia procesul cu următoarea iterație a buclei de control.

2.3.3 Transformarea PIM în PSM

PIM poate fi transformat în PSM prin intermediul următoarei transformări :

- Fiecare componentă autonomă se transformă într-o clasă care contine trei componente WSDM descrise în Figura 6. Fiecare dintre aceste componente este de fapt o clasă având metodele date.
- Fiecare legătură autonomă se transformă într-un sistem bazat pe mesaje SOAP sau WS-A. Pentru aceasta transformare, fiecare terminal al unei legături autonome generează o vecinătate SOAP care va utlizata pentru transmiterea şi recepționarea de mesaje.

2.3.4 Codul executabil

PSM se poate transforma parțial în cod executabil prin utlizarea Eclipse & Test Performance Tools Platform Project (TPTP) [BC-02] care permite generarea de cod WSDM pe baza modelelor ce descriu funcționarea resurselor. Din moment ce TPTP conține deja metode de genereare a unui cod WSDM corect, el a fost folositpentru generarea acestui tip de cod. Pentru implementarea specificațiilor WSDM s-a utilizat programul Apache Muse [AB-05b].

Generarea codului trebuie să creeze şi elementele programului Apache care permit lansarea de aplicații bazate pe J2EE. Aceste elemente sunt reprezentate de fişierul services.xml care conține descrierea tuturor celor trei resurse, fişierele WSDL, unul pentru generator şi unul pentru resursele de gestiune şi execuție. În lucrarea [AW-00b] este prezentată o soluție de implementare a sistemului pentru virtualizarea serverului.

2.5 SISTEME AUTONOME ADAPTIVE

Aşa cum s-a arătat în paragraful 2.2 arhitectura de baza împarte orice sistem autonom în 7 module: senzori, filtre, estimatori, coordonatori, modele si identificatori de modele, markeri de decizie, estimatori şi actuatori. Fiecare modul este implementat ca şi o componentă care poate fi încărcată şi descărcată dinamic la o solicitare din exterior. Coordonatorul, modelul, şi estimatorul controlerului care preia datele filtrate foloseşte cele 4 module pentru a lua o decizie şi trimite această decizie mai departe actuatorilor pentru controlul sistemului gestionat. Toate comunicațiile între module au loc APIs predefinite, astfel încât oricare filtru poate comunica cu orice senzor sau controler.

Modulele de comunicații includ de asemenea mecanisme de administrare a cererilor bazate pe sistemele de execuție în timp real. Alte exemple au fost utilizate pentru a controla accesul la resurse prin intermediul algoritmilor concurenți, cum ar fi creşterea siguranței întregului sistem autonom. Mai departe, cadrul principal foloseste un registru de componente care permite entităților externe să caute şi să găsească componentele disponibile într-o bibliotecă de componente.

În această accepțiune conceptuală, ciclul autonom de control este dezvoltat prin reconfigurarea buclei autonome, bazată pe nevoile şi schimbările din baza de infrastructura IT. În Fig. 2.16 este prezentată arhitectura sistemelor autonome adaptive organizată pe 3 niveluri. Săgețile din schemă reprezintă fluxurile de date de la entitățile gestionate prin senzori (informații monitorizate), filtre sau reconfigurarea entității gestionate. Nivelul inferior este compus din sistemul gestionat care poate fi un calculator, un grup de servere, o aplicație reală sau virtuală. Nivelul intermediar se compune din modulul ciclului de control autonom care poate fi afectat de perturbații generate de numărul utilizatorilor care solicit cereri pe servere. În final, nivelul superior este structura de adaptare care adună informațiile despre execuția ciclului autonom de control şi dacă este necesar chiar şi de la sistemul de gestionare.

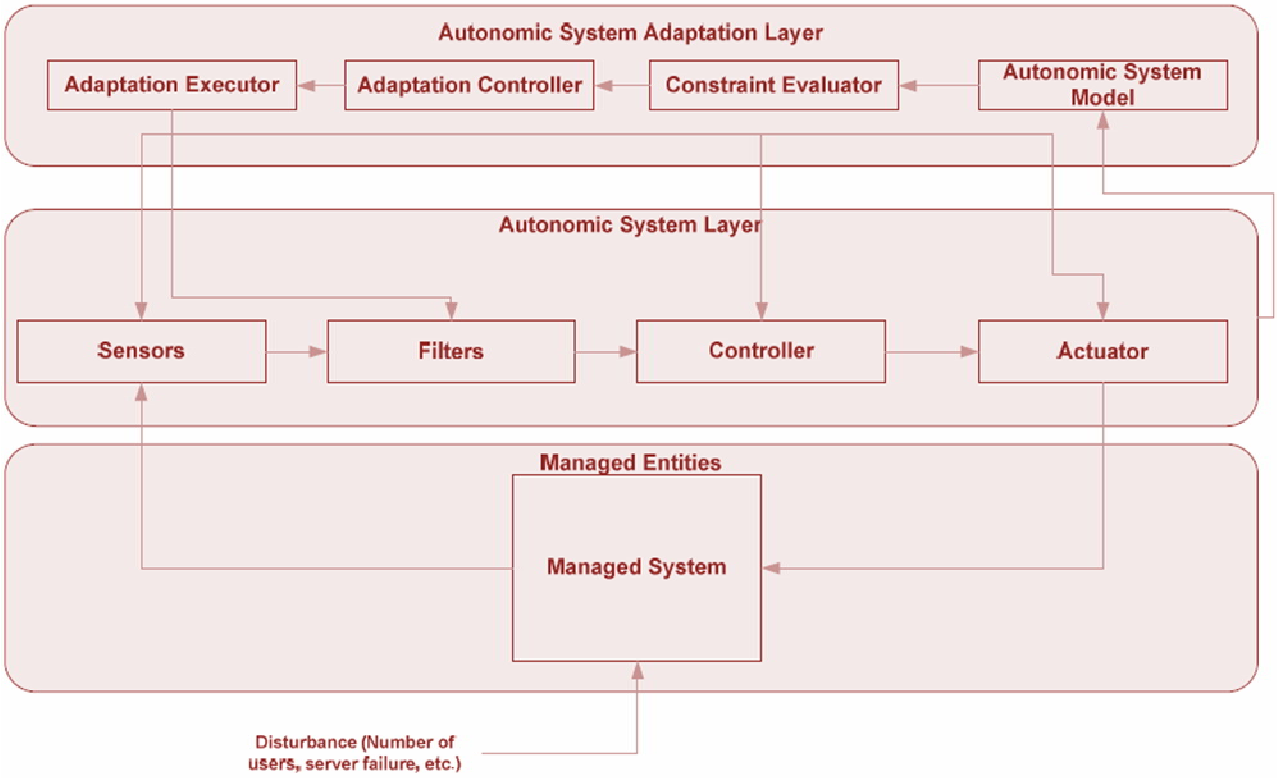


Figura 2.16 Sistemul autonom adaptiv

La acest nivel, se evaluează constrângerile, se iau deciziile de adaptare bazate pe evaluare şi se execută decizia adaptării prin reconfigurarea sau înlocuirea uneia sau mai multor module din cadrul sistemului autonom.

Pentru a realiza un ciclu de control autonom adaptabil sunt necesare perfecționări ulterioare pentru arhitectura propusă care să permită reconfigurarea dinamică în timpul functionării ciclului de control. Aceste mecanisme sunt utilizate pentru a salva starea unui modul, astfel încât să poata fi încărcată la cerere într-un modul de rezervă.

2.4.1 Arhitectura sistemelor autonome adaptive

Un sistem autonom adaptiv e necesar atunci când sistemul de bază gestionat se modifică în aşa fel încât modulele sistemului autonom nu mai caracterizează în totalitate sistemul inițial şi starea lui. Spre exemplu adăugarea unui server într-o grupare de servere necesită adăugarea unui nou senzor pentru a monitoriza noul server de date, şi reconfigurarea componentelor astfel încât să se obțină date de la senzor. În continuare, modelul care face parte din controler, necesită reconfigurarea, deoarece acum există mai multe servere. Aceste noi schimbări nu pun mari probleme, deoarece adăugarea senzorilor şi reconfigurarea controlerului pot fi facute fără a afecta în mod direct ciclul de control.

Există şi cazuri mai complicate, unde modificările în cadrul sistemelui autonom necesită mecanisme mai complicate pentru a rezolva reconfigurarea ciclului de control. De exemplu, dacă sistemul autonom este responsabil cu aprovizionarea unui grup de servere de aplicații şi numărul de utilizatori care accesează sistemul este o funcție liniară, atunci controlerul poate fi bazat pe un simplu prag şi când acest prag este depăşit, sunt adăugate noi servere. Dacă utilizatorii care accesează sistemul nu sunt liniari, dar sunt bazati pe o distributie gaussiană, atunci un controler bazat pe un filtru Kalman ar fi mai potrivit[AT-02].

Distribuția utilizatorilor sistemului se poate modifica în timpul rulării programului. De exemplu în cazul unei aplicații bancare, gradul de utilizare a sistemului reprezintă momentul când se termină ziua de muncă, sau la sfârşitul lunii, când oamenii sunt remunerați. În asemenea cazuri sistemul autonom ar trebui reconfigurat pentru a se adapta unei noi distribuții de utilizatori. Acesta poate fi bazată fie pe nişte valori predefinite, cum ar fi un timp sigur, când modelul se modifică sau bazat pe detecția schimbărilor în distribuția utilizatorilor.

Când asemenea schimbări sunt necesare, informațiile legate de starea controlerului sunt salvate, controlerul este oprit, înlaturat, şi se creeaza un nou controler bazat pe starea vechiului controler. Valorile finale folosite de ciclul de control computerizat autonom ca şi parametrii ai modelului sunt conservați şi reprezintă valorile inițiale pentru noul model. În timp ce se execută schimbarea, restul ciclului de control funcționează în continuare, orice modificare trimisă către actuator este executată, iar senzorii şi filtrele continuă să îşi îndeplinească funcțiile. Pentru a obține acest deziderat, modelul de încredere trebuie folosit pentru a se asigura că ciclul de control nu eşuează în timp ce câteva din componente se află într-o stare de inconsistență. Obiectivul celor două sisteme autonome este acelaşi: asigurarea timpului de răspuns corespunzător pentru aplicațiile web date şi utilizarea unui număr minim de servere de aplicații.

Acest exemplu poate fi direcționat către modulele din Fig. 2.2, după cum urmează. Modelul sistemului autonom se adaptează periodic bazându-se pe legile matematice care descriu variația numărului de utilizatori care folosesc serverul de aplicații. Aceste informații sunt furnizate mai departe evaluatorului de constrângeri. Când evaluatorul determină că distribuția utilizatorilor nu mai este liniară, semnalizează controlerul de adaptare cu noile informații despre distribuție(pentru acest caz o distribuție Gaussiana). În acest moment controlerul poate să ceară ca executorul adaptării să reconfigureze ciclul de control prin modificarea controlerului într-un controler bazat pe filtrul Kalman.

În continuare poate să ceară ca să fie inițializat controlerul pe baza informațiilor actuale despre distribuțiile furnizate de către evaluator. În timp ce controlerul este înlocuit, senzorii continuă să achiziționeze date şi orice informație în legătură cu serverele noi este adăugată modelului de îndată ce noul controler a fost instalat. Pentru a ajunge la aceasta funcționare, trebuie adăugat un mecanism de sesizare care să comunice cu restul modulelor că o componentă este izolată sau modificată.

2.4.2 Ciclul adaptiv de control

Aşa cum am prezentat în paragraful 2.4.1, nivelul de adaptare este alcătuit din 4 componente care formează un ciclu secundar responsabil cu monitorizarea, evaluarea şi modificarea ciclului de control.

Acest ciclu de control este similar cu cel utilizat la nivelul sistemului autonom, specializat pentru fiecare aplicație. În acest caz este aplicată o metodă de identificare a sistemului, pentru modelul sistemului care determină o modificare a structurii controlerului cu scopul de a adapta întregul sistem computerizat autonom la procesul real pe care îl controlează. Mecanismul astfel obținut poate fi dezvoltat în continuare, prin introducerea conceptului maşină-învațare ca instrument virtual pentru construcția modelelor în direcția de evoluție specificată a sistemului autonom specializat.

2.4.3 Componentele sistemului de control

Teoria controlului foloseşte o arhitectură de referință simplă şi general aplicabilă. Principalul obiectiv al acestei arhitecturi este să asigure componentele pentru a manipula un sistem obiectiv pentru a atinge obiectivul prevăzut dorit. Fig. 2.17 prezintă această arhitectura de referința [DI-08].

Componenta care manipulează sistemul obiectiv este denumită un “controler”. Sistemul obiectiv este resursa computerizată, controlerul este un conducător autonom şi obiectivul controlat este parte din componenta cunoaştere, referitor la arhitectura computerizată autonomă.

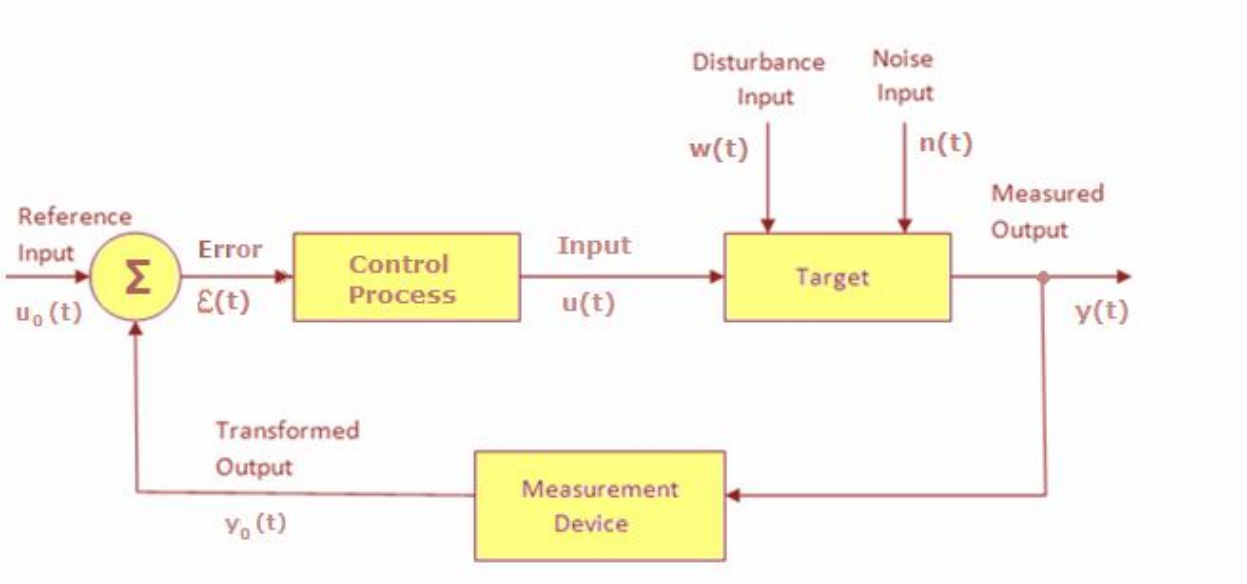


Fig. 2.17: Schema bloc a unui sistem control

Celelalte componente ale arhitecturii de referință sunt:

- O intrare de referinta $u_0(t)$, care reprezintă valoarea dorită a mărimii de ieşire (măsurate) sau o transformarea a acesteia. Exemplele includ timpul de răspuns dorit, utilizarea CPU, şi timpul de coborare într-un cadru de încredere.
- O eroare de control ε(t), care este diferența dintre intrarea de referință şi ieşirea transformată. Eroarea de control este folosită ca intrare pentru componenta controlerului.
- O intrare de control u(t), care este determinată de componenta controlerului şi folosită la influențarea comportamentului sistemului obiectiv. Intrarea de control va influența uneori sistemul obiectiv prin intermediul unui actuator. Exemplele includ numărul de servere într-o aşezare împrăştiată de desfăşurări şi parametrii de acordare pentru aplicația software.
- O ieşire măsurată y(t), care este o caracteristică măsurabilă a sistemului obiectiv. Exemplele includ utilizarea CPU şi timpul de răspuns al aplicației.
- Intrarea zgomot n(t), care poate fi orice afectează modificările la ieşirea măsurată produsă de către sistemul obiectiv.
- Ieşirea Factorului Perturbator d(t), care este orice modificare care afectează modul în care intrarea de control influențează iesirea măsurată. Exemplele includ un proces backup sau un proces virus-scan ce acționează pe durata măsurătorii.
- Sistemul Obiectiv, care este sistemul computerizat ce urmează a fi controlat.
- Unitatea de Măsurare, care transformă ieşirea măsurată astfel încât ea să poată fi comparată cu intrarea de referință. Un exemplu de transformare este filtrarea ieşirii măsurate pentru a reduce zgomotul stocat şi filtrarea trece jos pentru eliminarea perturbațiilor de înaltă frecvență

- Controlerul, care este componenta care determină setarea intrării de control necesare pentru a atinge valoarea dorită după cum a fost setată de către intrarea de referință. Controlerul calculeaza valorile pentru intrarea de control pe baza valorilor actuale, sau din trecut, ale erorilor de control.

Să considerăm spre exemplificare o distribuție spațială de aplicații (în jurul serverului Wiki), asa cum se exemplifică in Fig. 2.18 .

Administratorul sistemului poate dori ca aceste sisteme să lucreze cu un timp de răspuns garantat mai mic de 1,5 secunde luând în considerare că soluția de alocare cu mai multe resurse este o soluție costisitoare. În acest caz, iesirea măsurată este timpul de răspuns şi intrarea de control este numărul de servere în aplicațiile software împrăştiate.

Numarul de servere poate fi ajustat dinamic şi păstrează timpul de răspuns între limitele prescrise.

Exemple de perturbații sunt o gamă variată de cereri ale sistemului venind de la o varietate largă de clasa de useri, cât şi datorită diferitelor tipuri de cereri care conduc spre diferite căi de executare a aplicației.

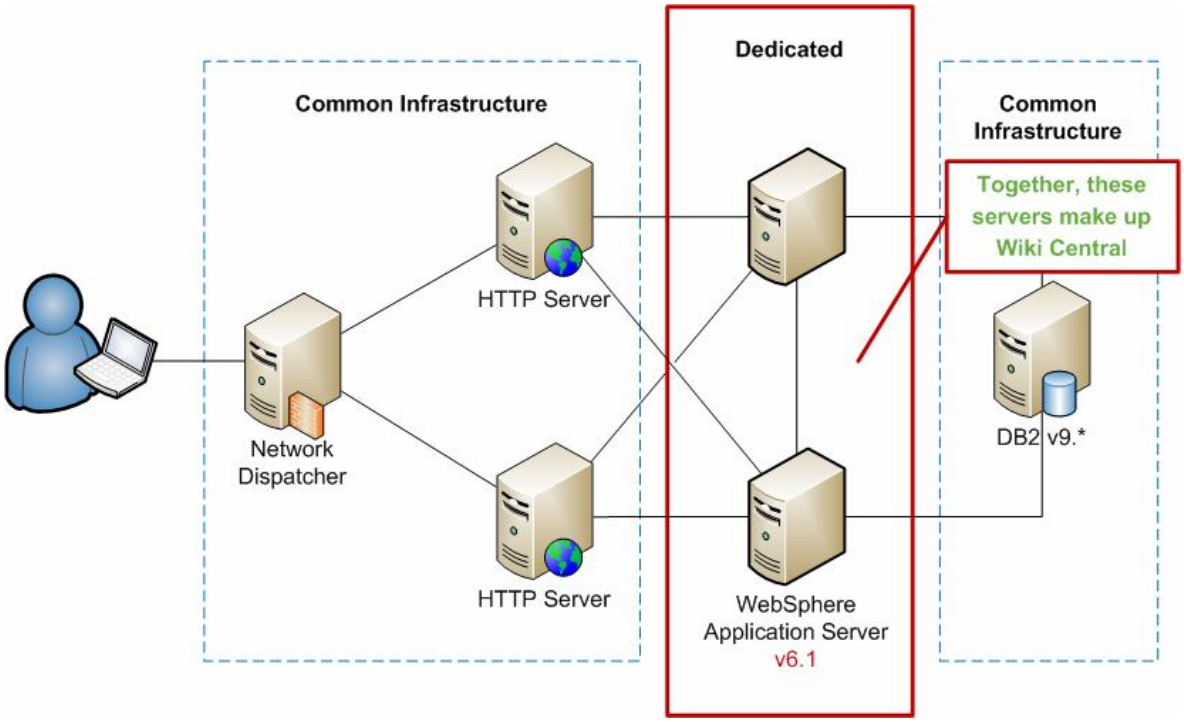


Fig. 2.18: Distribuția spațială de aplicații a unui Server Central Wiki

Într-o arhitectură autonomă, atenția este îndreptată spre specificarea şi construirea de componente care interopereaza bine în scopul executării țintelor de conducere. De exemplu, arhitectura computerizată autonomă este concentrată pe efectori şi a dezvoltat conceptul de conducere automată (autoconducere), bazată pe interfețe şi pe protocoale specifice.

Influența şi impactul teoriei de control este pe analizarea şi dezvoltarea componentelor şi algoritmilor care, atunci când sunt implementați, permit sistemului care rezultă să atingă obiectivele de control, să păstreze sistemul stabil şi să-l stabilizeze repede ca răspuns la perturbații.

2.4.4 Modelul sistemului autonom

Modelul sistemului autonom adună şi înmagazinează informații despre execuția ciclului de control. Pentru a realiza acest lucru modelul are cunoştiințe despre toate modulele din cadrul nivelului sistemului autonom. Aceste informații pot să conțină starea ciclului de control şi pragurile în care ciclurile de control sunt valide pentru modelele de bază.

În exemplul dat această informație ar include valorile distribuției utilizatorilor pentru care ciclul autonom de control este valid. În continuare modelul poate, de asemenea, adăuga informații despre sistemul de bază care este gestionat de ciclul de control. Aceste informații pot fi primite de la senzorii sistemului autonom. În exemplul dat, aceste date ar descrie distribuția curentă a utilizatorilor.

Valoarea poate fi obținută fie direct de la senzori, fie de la modelul entității de bază , ambele făcând parte din nivelul sistemului autonom.

2.4.5 Evaluatorul de constrângere

Responsabilitatea evaluatorului de constrângere este de a folosi logica predefinită pentru a determina dacă logica activității necesită schimbări la nivelul sistemului autonom.

Această componentă foloseşte o regulă a motorului reactiv pentru a determina dacă constrângerile sunt în afara limitelor valide, şi de a declanşa un eveniment în controlul de adaptare.

Evaluatorul descris în contextul acestei teze primeşte date de la un modul de test care evaluează distribuția în timp a numărului de utlizatori şi care este bazată pe binecunoscutul test x^2. Evaluatorul verifică dacă distribuția curentă a utilizatorilor este una din distribuțiile acceptate de controler. Dacă distribuția curentă este acceptată, nu va fi făcută nici o modificare; dacă distribuția nu este acceptată, atunci un semnal este transmis controlerului pentru ca acesta să ceară modificarea controlerului cu un controler valid pentru noua distribuție de utilizatori.

În exemplul luat în această lucrare, aceste evenimente au loc dacă numărul utilizatorilor are o distribuție lineară în timp. În acest caz testul x^2 va da greş în furnizarea unui rezultat care să arate distribuția numărului de utilizatori în timp ce regulile evaluatorului vor verifica dacă valorile consecutive ale distribuției utilizatorilor urmăresc o lege lineară.

2.4.6 Controlerul pentru adaptare

Controlerul pentru adaptare primeşte o notificare de la evaluatorul de constrângeri dacă ciclul autonom de control mai este valid pentru gestionarea sistemului, dar şi informații referitoare la parametrii care depăşesc limitele validității. Folosind această informație, în continuare, controlerul de adaptare alege actiunea care trebuie luată pentru a se asigura că ciclul de control este valid pentru sistemul gestionat. Această acțiune poate fi ceva simplu, cum ar fi modificarea parametrilor unei componente, sau complicată, cum ar fi înlocuirea componentei cu una nouă. În exemplul dat modulul controlerului va trebui înlocuit cu un controler care este valid pentru distribuția numărului de utilizatori dat. Pentru a îndeplini aceasta, controlerul de adaptare înmagazinează în simple baze de date proprietățile diferitelor module pe care le poate desfăşura la fel de bine ca şi limitările acestor module.

Pentru a perturba ciclul de control autonom cât de puțin posibil se poate folosi un controler de adaptare ierarhic similar cu cel sugerat în [5]. Acest controler ierarhic va încerca, pentru inceput, să vadă dacă simpla modificare a parametrilor componentei este suficientă pentru a aduce controlerul autonom înapoi la validitate. Dacă modificarea parametrilor nu este suficientă, numai atunci, va declanşa o cerere de înlocuire a componentei.

2.4.7 Executorul de adaptare

Rolul lui este de a efectua modificările asupra ciclului autonom de control. Bazat pe o cerere venită de la controlerul de adaptare, executorul de adaptare declanşează execuția necesară pentru modificarea sau înlocuirea componentei ciclului de control de calcul autonom. Executorul foloseşte APIs predefinite pentru modificarea parametrilor componentei, ca să fie independentă de la punerea în aplicare a controlerului. Executorul preia toate valorile apropiate, pentru a se asigura că partea de ciclu de control autonom nu este modificată sau înlocuită şi nu este afectată de schimbările din ciclul de control.

Executorul este de asemenea responsabil de a informa componentele ciclului de control când componenta modificată este activă şi gata să fie folosită din nou. Dacă este necesară funcția de răspuns de la executor la modelul sistemului autonom, executorul este cel care receptionează modificările şi conține logica necesară pentru modificarea modelului. În exemplul dat, executorul va opri controlerul autonom, va menține starea lui pentru o viitoare utilizare şi va înştiința filtrele şi actuatoarele de invaliditatea controlerului. Pe urmă va încărca noul controler autonom, îl va inițializa cu starea existentă a controlerului indepărtat şi va înştiința filtrele şi actuatorii că noul controler este valid şi că funcționează.

Odata ce ciclul autonom de control este din nou într-o stare functionară, executorul de adaptare va modifica, de asemenea, modelul de sistem autonom, pentru a putea specifica noul tip de controler utilizat şi parametrii acestora.

2.4.8 Salvarea stărilor

Mai întâi de toate, ciclul de control adaptiv trebuie să păstreze starea ciclului de control autonom, în timp ce sunt efectuate modificări asupra acestora. Pentru ca acest lucru sa fie îndeplinit, ciclul de control adaptiv va necesita cunoştiinţe despre ciclul autonom care rulează.

În acelaşi timp procesul de salvare de stare ar trebui sa necesite o perioada scurta de timp pentru a opri ciclul de control pentru o perioada prea mare.

Exista două metode pentru a salva starea sistemului în timpul funcţionării sale, cum e descris în [10]: salvarea punctelor de control şi incrementarea stării salvate.

Salvarea punctelor de control se referă la o salvare periodică a întregii stări a sistemului în anumite puncte critice în timp, înainte sau după execuţia unor evenimente importante.

Această metodă creează, în general, o suprasolicitare, deoarece întreaga stare a sistemului este salvată de mai multe ori; oricum, în cazul buclei adaptive, doar ultimul punct va fi nevoie să fie salvat. În acelaşi timp, ciclul de control nu se execută în mod continuu, şi astfel salvarea stării poate avea loc în timp ce ciclul autonom este într-o perioadă de repaus a execuţiei.

Incrementarea salvării stării se referă numai la salvarea modificărilor care au avut loc de la starea iniţiaăa. Acest rezultat aproximativ nu conţine atâtea date, deci necesita mai puţin timp. Totusi, în majoritatea cazurilor de încărcare a stării, etapa de încărcare durează mai mult deoarece întreaga stare trebuie construită pas cu pas.

În cazul sistemului autonom, starea unei componente este reprezentată de valorile parametrilor folosiţi de componentă, la fel de bine ca şi cunoştiinţele despre componentele cu care trebuie să comunice. Din cauza felului cum este executat ciclul de control, metodologia stării persistente poate fi o combinaţie a două metode descrise. Fiecare componentă îşi salvează întreaga stare periodică de-a lungul unei perioade de repaus. Din moment ce starea iniţială a componentei nu este importantă, punctul de control anterior poate fi înlocuit cu noua stare. O a doua fază a stării persistente este activată când componenta trebuie înlocuită.

În acest moment se face o salvare incrementală, care înmagazinează doar valorile care au fost modificate, de la punctul de control salvat. Starea este salvată sub forma de perechi nume-valoare, în orice depozit permanent (fişier, bază de date), şi se presupune că executorul de adaptare ştie cum să traducă din variabilele unei componente în variabilele înlocuitoarei. Când starea este încărcată în noua componentă, doar variabilele stării cerute de către aceasta sunt încărcate, prin verificarea existenţei lor, prima data în salvarea incremenală şi pe urmă în punctele de control salvate.

2.4.9 Executia ciclului

2.4.10

În timp ce una sau mai multe componente sunt oprite şi înlocuite, restul ciclului de control autonom trebuie să funcționeze cât mai bine cu putință.

Aceasta înseamnă, de exemplu, că în timp ce controlerul este înlocuit, senzorii şi filtrele continuă să îşi îndeplinească sarcinile şi că actuatorii îndeplinesc orice cerere din lista de aşteptare de la controlerul anterior. Pentru a îndeplini această funcționabilitate, toate componetele trebuie să recepționeze informații în legătură cu starea componentelor care comunică, având la bază cunoştiințele despre starea tuturor componentelor, să trimită sau să primească date de la acestea.

Este implementat un mecanism de înştiințare, care să permită executorului de adaptare să înştiințeze componentele din ciclul autonom de control. În continuare, câteva componente s-ar putea să utilizeze şiruri în care să înmagazineze informațiile până când acestea pot fi livrate către componenta nouă adaugată.

Decizia în legătură cu implementarea sau nu a şirurilor, este componenta dependentă, ca de exemplu, unui senzor i se poate cere să transmită ultimele valori, în timp ce un actuator va fi nevoit să pună toate acțiunile într-un şir, pentru a îmbunătăți modelul de la baza sistemului.

2.5CONCLUZII ŞI CONTRIBUȚII

Calculatoarele autonome tind să înglobeze structuri automate în sistemele de management informatic, astfel încât să se poată adapta schimbărilor de configurație, protecție şi reconfigurarea resurselor disponibile pe parcursul funcționării. Una din direcțiile actuale de cercetare consideră calculatoarele autonome ca reprezentând un sistem de control adaptiv care rezolvă limitările impuse de utilizarea optimală a resurselor disponibile ca urmare a unei solicitări externe. În cadrul acestui capitol au fost tratate următoarele probleme principale:

- Analiza şi prezentarea stadiului actual de dezvoltare a mediului de calcul pentru sistemele autonome cu funcționare în timp real.
- Prezentarea arhitecturii sistemelor de calcul autonome şi dezvoltarea buclelor de control şi răspuns în sistemele computerizate autoconduse.
- Elaborarea unei arhitecturi generice pentru sistemele numerice de control autonome. În această arhitectură, s-a considerat că resursa de bază poate fi un sistem computerizat.
- S-a demonstrat că entitățile software care asigură funcțiile sistemelor de calcul autonome aparțin unei platforme complexe care găzduieşte un proces de calcul performant şi care devine dificil de gestionat dacă funcționează pe baza unui sistem centralizat.

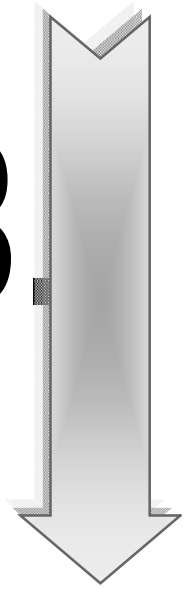
- Analiza componentelor de bază ale sistemelor de control din echipamentele de calcul autonom şi specificarea funcțiilor principale pe care le îndeplinesc.
- Adaptarea conceptului modelării şi optimizării funcției de bază a componentelor, ca soluție de implementare în timp real a unor noi componente interschimbabile.
- Se abordează arhitectura sistemelor autonome reconfigurabile bazate pe modelul „model – platform" independentă şi specifică, ca aplicație multistivă care se autooptimizează.
- Se demonstrează că performanțele aplicațiilor multistivă interactive pot fi exprimate ca obiective la nivel de servicii (Service Level Objectives). Scopul urmărit îl constituie evalările limitelor superioară/inferioară a timpilor de răspuns, pentru ieşirile fiecărui tip de tranzacție efectuată de sistem.
- Utilizarea serviciilor Web pentru infrastructura de comunicație care acționează atât ca interfață pentru client, cât şi pentru sistemul de gestionare a datelor.
- Având la bază evaluările anterioare, se analizează posibilitatea implementării unei arhitecturi de referințăcu funcționare în timp real, care să monitorizeze comportamentul global al mediului de calcul autonom, utilizând controlul adaptiv.

Contribuțiile originale aduse de autor la elaborarea acestui capitol pot fi rezumate astfel:

- Elaborarea unei teorii unitare şi adaptarea conceptului de „sisteme autonome de calcul" ca bază a managementului optimal al procesului de automatizare a infrastructurii informaționale actuale.
- Analiza mediului de calculinformațional autonom, ca bază a construirii sistemelor inteligentede management informatic cu aplicații industriale, pornind de la soluțiile dezvoltate in mediile academice.
- Prezentarea unei structuri hardware minimale, pentru calculatoarele numerice autonome, conținând: senzori, traductoare, controlere, actuatori, conectați la sisteme puternice de calcul, prin similtitudine cu echipamentele de contro inteligent automat cu funcționare în timp real.
- Elaborarea unei structuri generice pentru sistemele computerizate autonome, care reprezintă fundamentul utilizării teoriei de control automat, la structurarea conducerii echipamentelor numerice multiple, conectate la aceaşi unitate centrală de prelucrare.

- Analiza comparativă a diferitelor arhitecturi de sisteme numerice autonome cunoscute în literatura de specialitate, în vederea stabilirii soluției optimale, capabilă să se configureze pe structura existentă şi să se autoconfigureze pentru o mai bună gestionare a infrastructurii.
- Pornind de la conceptul modelării şi optimizării funcției de bază a componentelor, s-a propus o soluție novativă care permite generarea de componente independente, reutilizabile, capabile să preia funcțiile componentelor defecte.
- Analiza efectuată a demonstrat că pentru asigurarea unui flux de date neperturbat între elementele constituente şi sistemul de gestionare este necesar să se realizeza comunicația prin intermediul serviciilor Web.
- Pornind de la constatarea că infrastructura IT a devenit vitală pentru funcționarea societăților şi chiar a guvernelor, şi că erorile de perturbare sau comunicare pot genera rezultate catastrofale, s-a lansat ideea construirii unui mediu software complex capabil să: supravegheze, optimizeze, protejeze şi să adapteze infrastructura IT existentă printr-o autogestionare inteligentă a platformei.
- Elaborarea contribuțiilor specifice modelelor „platformă independentă" şi „platformă specifică", precum şi elementele necesare pentru generarea codului executabil, pornind de la diagrama stărilor care gestionează aceste modele.
- În scopul creşterii performanțelor la costuri accesibile s-a propus şi anallizat o arhitectură modernă utilizând structurile autonome computerizate adaptive, organizate ierarhic.
- Analiza caracteristicilor principale pe care trebuie să le indeplinească: Evaluatorul de constrângeri, Controlerul pentru adaptare, Executorul de adaptare ca şi celelalte blocuri funcționale a permis elaborarea unui „Banc de probă" care constituie suportul rezultatelor experimentale menite să confirme dezvoltările teoretice prezentate pe parcursul elaborării tezei de doctorat.

În final, considerăm că studiul inițiat în cadrul acestui capitol este capabil să prezinte o modalitate eficientă petru îmbunătățirea performanțelor unui sistem autonom de control capabil să se automodifice şi să se adapteze la schimbările de mediu. Prin adăugarea unui ciclu suplimentar de control automat, având ca obiectiv gestionarea sistemului de calcul autonom, se poate asigura reconfigurarea în timp real şi adoptarea unei noi bucle de control.



3. MODELE PREDICTIVE DE CONTROL PENTRU SISTEME AUTONOME DE CALCUL

Modelul predictiv de control (MPC) este un concept modern utilizat la monitorizarea proceselor industriale. Nucleul unui model predictiv de control îl reprezintă funcția de transfer a sistemului. Orice model predictiv de control al unui proces va trebui să fie capabil să estimeze viitoarele semnale de ieşire pe baza viitoarelor semnale de intrare şi a valorilor inițiale. Cu acest model de proces, viitorul comportament dinamic al unei instalații reale este analizat în cadrul unui orizont bidimensional N_2. Cu aceste ipoteze, semnalele de ieşire se utilizează pentru a optimiza performanțele criteriului în buclă deschisă şi pentru a calcula semnalul de intrare u(k) într-un orizont de control în N_u. În afara orizontului de control, semnalul de intrare u(k) rămâne constant. Semnalele de intrare calculate sunt achiziționate de sistem până când un nou set de semnale obținute dintr-o nouă măsuratoare va fi disponibil. Această procedură este repetată cu un nou orizont de predicție şi control şi se numeşte controlul orizontului îndepărtat. Strategia orizontului îndepărtat creează o regulă de control a buclei închise provenită din minimizarea buclei deschise originale. Pasul de minimalizare poate fi modificat astfel încât semnalele de intrare-iesire sau condițiile de stare, să poată fi luate în considerare la proiectarea controlerului.

3.1 ANALIZA MODELULUI PREDICTIV DE CONTROL

Tehnologia MPC a câştigat o mare popularitate în controlul proceselor industriale abordând problema optimizării parametrilor de proces în condiții fizice neprevăzute dar şi prin capacitatea sa de a controla sistemele cu variabile multiple.

3.1.1 Stategia modelului predictiv de control:

Modelul predictiv de control ,în mod normal dezvoltă următoarele trei idei:

1. Folosirea explicită a unui model pentru a urmări evoluția desfăşurării procesului pe o anumită perioadă de timp.
2. Calculul unei secvențe de control pentru a optimiza indexul unei performanțe.
3. O strategie pe termen lung, astfel încât în fiecare moment de timp, semnalele de control să estimeze secvența de date viitoare în condiții inițiale.

Strategia modelului predictiv de control este ilustrată în Fig. 3.1:

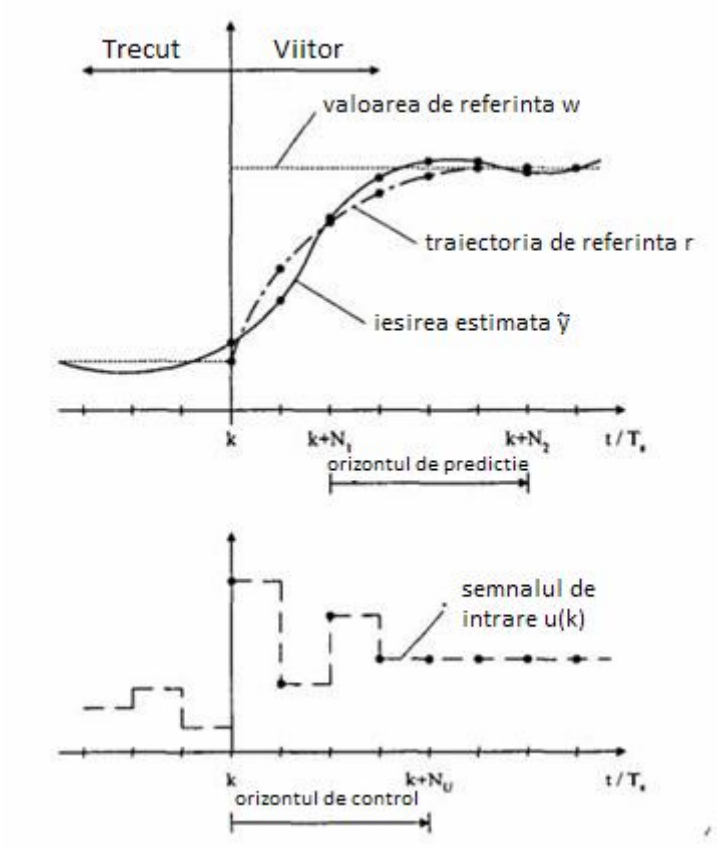


Fig. 3.1 Strategia modelului predictiv de control

1. Presupusele viitoare ieşiri $\hat{y}(t+k|t),\ k=1\cdots N$ pentru orizontul de predicție N sunt calculate la fiecare moment t folosind modelul procesului.Acestea depind de cunoasterea valorilor pana la momentul t(iesirile si intrarile trecute),inclusiv iesirea curenta(conditia initiala) y(t) si a viitoarelor semnale de control u(t+k|t), k=0…..N-1 care trebuie calculate (notatia x(t+y|t) exprimă valoarea lui x la momentul de timp t+k, calculata pornind de la valoarea corespunzătoare momentului de timp t)
2. Secventa viitorului semnal de control este calculata pentru a optimiza un criteriu de performanta, deseori pentru a minimiza erorile dintre o traiectorie de referinta si un presupus proces de iesire.De obicei efortul controlului predictiv este inclus in criteriul de performanță şi se regăseşte în costurile solicitate de prelucrare.
3. Numai actualul semnal de control u (t|t) este transmis procesului.La urmatorul moment de esantionare y(t+1) este masurat si este repetat pasul 1 si toate secventele sunt actualizate. În final semnalul de control u(t+1|t+1) este calculat folosind conceptul orizontului indepartat.

3.1.2 Avantaje şi dezavantaje

Cele mai importante avantaje ale controlului predictiv sunt:

- conceptele sunt intuitive si atractive pentru industrie
- pot fi folosite pentru a controla o mare varietate de procese,inclusiv cele ‘fara faza minima, cu intarziere mare de timp sau sisteme instabile in bucla deschisa
- poate face fata proceselor cu intrari si iesiri multiple la fel de bine ca si celor cu o singura intrare/iesire
- constrangerile procesului pot fi tratate prompt in cadrul procesului de optimizare
- usor aplicabile in procesele in care viitoarele semnale de referinta sunt cunoscute
- inițiază o tehnologie care permite extensii pentru viitor

Dezavantajul major îl constituie :

- cerinta unui model potrivit al procesului şi costul ridicat la prelucrare

Tehnicile MPC, cunoscute în literatura de specialitate, se deosebesc ca abordare prin diferite tipuri de modele şi funcții de performanță. Algoritmii care stau la baza dezvoltării modelelor predictive de control pot fi sistematizați astfel :

- Controlul algoritmic al modelului (Model Algorithmic Control), MAC, inițial denumit Model de Predicție Euristica de Control(Model Predictive Heuristic Control) MPHC. Acesta foloseste un model cu raspuns impuls, care este valabil numai pentru procesele stabile in bucla deschisa, si minimizeaza variatia erorii dintre iesire si traiectoria de referință, calculata pentru sistemul de ordin I.

- Controlul dinamic al matricei DMC(Dynamic Matrix Control). Acest algoritm este similar cu MAC dar foloseste un model cu raspuns treapta in locul unui model cu raspuns impuls. Metoda a fost extinsa pentru a include constrangerile de manipulare a intrarilor si a iesirilor folosind programarea patratica pentru a rezolva problema optimizarii constrangerilor, dand nastere la Matricea Dinamica de Control Patratica (Quadratic Dynamic Matrix Control),QDMC. Algoritmul DMC poate fi de asemenea derivat pentru un model general discret stare-spatiu.

- Metoda extinsa a cotrolului auto-adaptabil sau Modelul de control nelinear (Extended Prediction Self Adaptive Control), EPSAC, foloseste o functie de transfer discreta (transformata z) pentru a modela procesul. Se utilizează principiul de control al structurii calculat analitic (folosind o functie pătratică) performantă presupunând ca u(t) ramane constant din momentul t. Modelul procesului include de asemenea perturbatii masurabile.

- Controlul Previzibil Generalizat(Generalised Predictive Control), GPC, este o functie patratica performanta, având ponderarea efortului de control, si o medie a miscarii regressive cu un model cu variabile exogene(ARMAX). De asemenea ofera o solutie analitica pentru un control optimal în absenta constrangerilor.

3.2 SISTEM PREDICTIV DE CONTROL NELINEAR BAZAT PE MODELUL WIENER

În aceast paragraf vom analiza, modelul Wiener standard, care este un model nelinear predictiv de control. El a fost încorporat într-o schemă de prelucrare în care nelinearitatea poate fi eficient aproximată cu o schemă de control predictiv lineară. Cele mai uzuale tehnologii MPC disponibile, se bazează pe un model liniar al procesului. Totuşi, când regiunea de operare este extinsă, neliniaritatea procesului nu poate fi ignorată şi performanţele MPC-ului liniar scad din cauza incapacităţii modelului liniar de a aproxima cu exactitate procesul real. În plus, există cazuri în care efectele neliniarităţii sunt importante ceea ce justifică utilizarea modelului neliniar predictiv de control. Aceasta include cel puţin două mari categorii de aplicaţii[BS-06a], [DC-06]:

1.1 Probleme de control a regulatorului, unde procesul este foarte neliniar şi este frecvent supus unor mari perturbaţii ridicate.

1.2 Probleme de servo control, unde punctele operaţionale se modifică frecvent şi cuprind o gamă largă de procese dinamice neliniare.

3.2.1 Analiza modelului Wiener

Cu introducerea unui model dinamic neliniar în algoritmul NMPC, complexitatea problemei estimării controlului creşte semnificativ [GG-05],[TH-03]. În acest sens multe cercetări au utilizat metoda calculului ortogonal pentru a discretiza ecuațiile diferentiale ale modelului in raport cu metoda optimizarii condițiilor [DB-05],[XW-06] sau tehnici succesive de liniarizare [JM-02] pentru aproximații.În aceste situații se justifică folosirea modelelor neliniare de control predictiv. Retelele neuronale recurente sunt cele mai populare 'modele' pentru estimările empirice.Dezvoltarea acestor modele este totusi dificila [DC-06]. În plus mai multe modele NARX pot fi folosite pentru a modela procesele neliniare. Determinarea ordinii si a structurii modelului a unui model NARX general este un exercitiu deficil chiar si pentru un sistem SISO si dificultatile sunt mai complexe pentru sistemele MIMO[CH-05].Modelele de serie Volterra pot fi folosite pentru a modela o larga categorie de sisteme neliniare,totusi aceste modele nu sunt economice in parametrii,si la randul lor,dificil de folosit pentru a modela sistemele MIMO [LL-03].

Modelele orientate pe blocuri au structuri speciale care le faciliteaza aplicarea in NMPC. Aceste modele sunt foarte apropiate de modelele liniare, deoarece ele constau intr-o serie de conexiuni de elemente liniare dinamice si un element static neliniar.

Daca blocul liniar dinamic ar fi urmat de o iesire statica neliniara, in acest caz, aceste modele sunt mentionate la modelel Wiener. În figura 3.2 este prezentat schematic modelul Wiener.

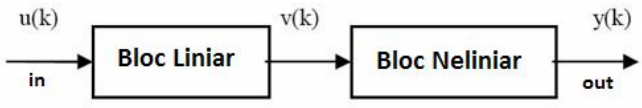


Fig. 3.2 Reprezentarea schematica a modelului Wiener

3.2.2.Formularea Wiener

Modelele Wiener au avantajul de a prezenta o neliniaritate statica care poate fi îndepartată eficient din problema controlului. Evident, aceasta neliniaritate nu poate fi ignorată, dar, structurând problema controlului aşa cum se exemplifică în Fig. 3.3, se simplifică mult optimizarea.

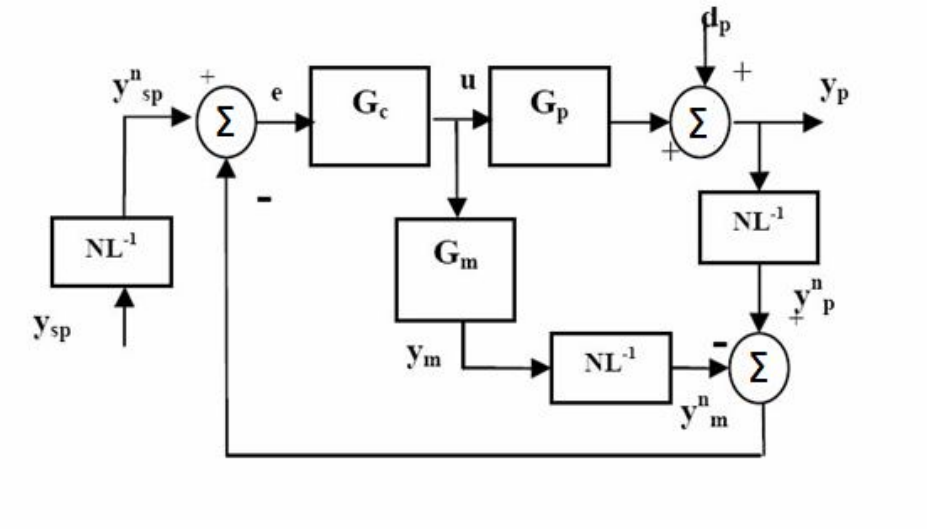


Fig. 3.3 Diagrama bloc al MPC – Wiener

După cum se observă în Fig. 3.3, neliniaritatea este tratată ca un simplu traseu static. Funcția inversă care este aplicată asupra iesirilor procesului, y_p si a modelului y_m, urmăresc liniarizarea controlului între intrarea u şi variabila intermediară x. Trebuie remarcat ca blocul G al modelului este alcătuit de fapt din 2 subsisteme, elementul liniar urmat de neliniaritatea statică NL^{-1}, care va anula prin inversare, neliniaritatea aplicată la ieşire (analizând doar elementul liniar al modelului pe acest traseu de prelucrare).

Se poate arăta, că daca se optează pe un criteriu pătratic si condiții inițiale nule, problema optimizarii in schema propusa WMPC rămâne o problemă de utilizare a unui program pătratic de calcul (QP) foarte convex[AC-05], care poate fi rezolvat eficient, apelând la algoritmi existenți. Pentru blocul liniar, o descriere a vectorilor de stare poate fi exprimată analitic după cum urmează:

$$x(k+1) = Ax(k) + Bu(k) \tag{3.1}$$

$$v(k) = Cx(k) + Du(k) \tag{3.2}$$

Dacă vectorul de stare in timp real si viitorul comportament al variabilei sunt presupuse a fi cunoscute, ele pot fi scrise într-o matrice în care identificăm:

- *Vectorul iesirilor pentru modelul liniar:*

$$V(k)=[v^T(k+1),v^T(k+2),....,v^T(k+p)]^T \quad (3.3)$$

- *Vectorul cu variabile reglabile:*

$$U(k)=[u^T(k),u^T(k+1),...,u^T(k+m)]^T \quad (3.4)$$

- *Vectorul de ieşire al modelului:*

$$Y(k)=[y^T(k+1),y^T(k+2),...,y^T(k+p)]^T \quad (3.5)$$

unde m şi p sunt orizonturile de control şi predicție. Ieşirea estimată v pentru modelul liniar poate fi scrisă:

$$v(k+1)=CAx(k)+CBu(k)+Du(k+1) \quad (3.6)$$

$$v(k+2)=CA^2x(k)+CABu(k)+CBu(k+1)+Du(k+2) \quad (3.7)$$

rezultă:

$$V(k)=\beta U(k)+\xi x(k) \quad (3.8)$$

unde parametrii β şi ξ se calculează cu relațiile:

$$\beta = \begin{bmatrix} C^T B & D & 0 & \cdots & 0 \\ C^T AB & C^T B & D & \cdots & \\ C^T A^2 B & C^T AB & C^T B & \cdots & \\ \vdots & \vdots & \vdots & \ddots & \\ C^T A^{p-1} B & C^T A^{p-2} B & C^T A^{p-3} B & \cdots & C^T A^{p-m} B \end{bmatrix} \tag{3.9}$$

$$\xi = \begin{bmatrix} C^T A \\ C^T A^2 \\ \vdots \\ C^T A^{p-1} \end{bmatrix} \tag{3.10}$$

În final, problema optimizării pentru formularea WMPC de mai sus constă în calculul funcției:

$$\min_{u(k),u(k+1|k),\cdots,u(k+M-1|k)} J = H[v(k+p|k)] + \sum_{J=0}^{p-1} \mathrm{L}\left[v(k+j|k), u(k+j|k), \Delta u(k+j|k)\right] \tag{3.11}$$

unde $u(k+j|k)$ este intrarea u(k+j) calculată din informația disponibilă la momentul k, $v(k+j|k)$ este ieşirea modelului liniar calculată din informația disponibilă la momentul k, iar

$$\Delta u(k+j|k) = u(k+j|k) - u(k+j-1|k)$$

Funcțiile H şi L pot fi alese corespunzător pentru a îndeplini o mare varietate de obiective, inclusiv minimizarea costului total de proces.Totuşi, optimizările economice pot fi efectuate de un sistem de calcul performant care determină punctele de reglare corespunzătoare pentru controlerul NMPC. În acest caz este important să luăm în considerare funcțiile pătratice L, H care pot fi calculate cu relațiile:

$$\begin{aligned} L(v,u,\Delta u) = &\left[v(k+j|k) - y_s(k)\right]^T Q\left[v(k+j|k) - y_s(k)\right] \\ &+ \left[u(k+j|k) - u_s(k)\right]^T R\left[u(k+j|k) - u_s(k)\right] \\ &+ \Delta u^T(k+j|k)\ S\ \ \Delta u(k+j|k) \end{aligned} \tag{3.12}$$

$$H(v) = \left[v(k+p|k) - y_s(k)\right]^T Q\left[v(k+p|k) - y_s(k)\right] \tag{3.13}$$

$$y_s(k) = NL^{-1}(y_{sp}) - \hat{d}(k) \tag{3.14}$$

$$\hat{d}(k) = NL^{-1}(y_p) - v(k|k) \tag{3.15}$$

Semnalul $u_S(k)$ reprezintă starea de echilibru țintă pentru u, iar Q, R, S sunt matrici de ponderare pozitiv definite. Principalii parametrii de reglaj ai controlerului sunt aşadar: m, p, Q, R, S. Pentru acest caz particular, problema optimizarii poate fi redusă la analiza unei probleme de programare pătratică standard.

3.3 MODELUL PREDICTIV DE CONTROL NELINIAR ÎN SPAȚIUL MULTIDIMENSIONAL

Reprezentarea cea mai cunoscută a MPC o reprezintă Controlul Previzibil Generalizat (Generalised Predictive Control), GPC, care se bazează pe modelul liniar în timp discret al funcției de transfer.

O reprezentare stare-spatiu a GPC poate fi gasita in [8]. Ambele rerprezentări pot fi utilizate numai pentru procesele liniare invariante in timp. Un mare avantaj este solutia analitica a problemei minimizarii in absența conditiilor inițiale. În cazul specificării conditiilor inițiale, GPC conduce la utilizarea unui program patratic, pentru care exista algoritmi numerici rapizi si foarte eficienti.

Pentru aplicații de mare precizie modelele de proces neliniare, trebuie sa fie utilizate pentru estimare [CL-00b]. În general această observație conduce la utilizarea unui program non-convex neliniar,care este dificil de rezolvat deoarece: cheltuielile de calcul pentru programul neliniare sunt mult mai ridicate decat cele pentru programul patratic. Aceasta, restrictioneaza aplicarea modelului proceselor neliniare la analiza proceselor mai lente.

Un al doilea dezavantaj este faptul ca programele neliniare non-convexe au cateva minime locale. Prin urmare trebuie aplicate metodele de obtinere a unui minim global (care cresc cheltuielile de calcul si mai mult).

Această lucrare introduce un concept nou de model predictiv de control, care combină avantajele amândurora, modelelor proceselor de predicție liniare şi neliniare. Modelul procesului se presupune a putea fi separabil într-o parte liniară dinamică cu o cale de răspuns neliniară. Acest tip de model de proces se numeşte sistem cu neliniaritate izolată.

Ca şi în toate conceptele de estimare a controlului, traiectoria de referință este cunoscută în orizontul de predicție. Calea de răspuns neliniar este liniarizată de-a lungul acestei traiectorii. Sistemul care rezultă este liniar, dar variabil în timp şi este folosit pe post de model predictiv. Acuratețea sa este mai bună decât cea a modelelor liniarizate în jurul unui singur punct operațional, pentru că neliniaritatea este luată în calcul de-a lungul traiectoriei complete. Cheltuielile de calcul pentru modelul liniar variabil în timp sunt similare cu cele necesare pentru modelele liniare invariante în timp. Fără eforturi considerabile, problema minimizării poate fi rezolvată analitic şi nu este necesară aplicarea algoritmilor numerici de optimizare. Când includem condițiile, rezultatul problemei optimizarii este un program pătratic care poate fi rezolvat cu aceeaşi algoritmi eficienți ca şi GPC. Cea mai mare acurateţe a modelului proceselor neliniare, este combinată cu abilitatea de a aplica tehnicile programării pătratice pentru optimizarea online.

3.3.1 Modelul procesului

În cadrul acestei analize, vom considera un sistem SISO(o singura intrare şi o singură ieşire). O extensie către sistemele cu intrări/iesiri multiple, este posibilă fără o creştere exagerată a complexității. Sistemul SISO considerat este descris de un model de stări de spațiu neliniar în timp discret de grad N cu o neliniaritate izolată NL. Semnalele de intrare $x(k+1)$ şi de ieşire $y(k)$ pot fi modelate analitic prin sistemul de ecuații[KG-05].

$$\begin{cases} x(k+1) = A * x(k) + b * \text{EMBED Equation.3}\, u(k) + K * NL(y(k)) \\ y(k) = c\, \text{EMBED Equation.3}\, * x(k) + d * \text{EMBED Equation.3}\, u(k) \end{cases} \quad (3.16)$$

Schema bloc a sistemului de prelucrare este prezentată în Fig 3.4.

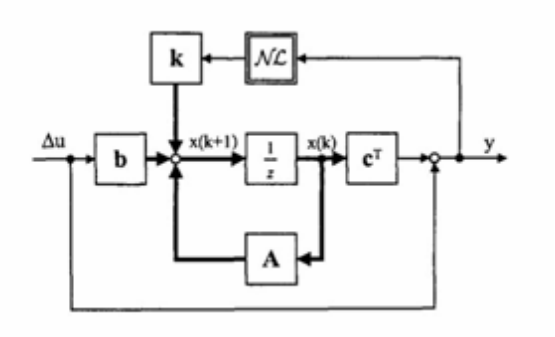


Fig. 3.4 Schema sistemului de prelucrare

Matricile sistemului A,b,c şi k sunt constante şi de aceleasi dimensiuni . Vectorul k descrie cuplajul neliniarității în cadrul sistemului. Δu reprezintă incrementarea semnalului de la intrare între două momente de esantionare. Orice sistem cu stări de spațiu având un semnal de intrare u (în loc de Δu), poate fi transformat cu ecuatia (3.16) adăugând variabila de stare adițională u(k-1). Operatorul de diferențiere Δ este definit astfel :

$\Delta = 1 - z^{-1}$ (z^{-1} este operatorul de intarziere cu un pas).

Matricile sistemului trebuie cunoscute, variabilele de stare se presupun a fi măsurabile şi neliniaritatea NL poate fi necunoscută. Dacă neliniaritatea şi variabilele de stare nu sunt disponibile, poate fi aplicat un observator si un identificator[10] sistemelor din aceasta categorie. Pentru toate calculele care urmează, facem presupunerea că neliniaritatea este cunoscută şi mărimile observate respectiv stările reale sunt identice. Pentru a beneficia de avantajul unui calcul simplificat, modelul sistemului (3.16) este liniarizat de-a lungul traiectoriei de referință cunoscută r(k) şi a semnalului de ieşire y(k). Traiectoria de referință reprezintă semnalul de ieşire dorit al sistemului. Trebuie să fie cunoscut în orizontul de predicție de la pasul k la pasul $k+N_2$,unde N_2 reprezintă orizontul superior de estimare. Liniarizarea neliniarității izolate de-a lungul traiectoriei de referinta r(k) conduce la relața:

$$\mathrm{NL}(y(k)) \approx \mathrm{NL}(r(k)) + \frac{dNL}{dy}\Big|_{y=r(k)} * (y(k)-r(k)) =$$

$$= \mathrm{NL}(r(k)) + \frac{dNL}{dy}\Big|_{y=r(k)} *(c * x(k) + d * \Delta u(k) - r(k)) \qquad (3.17)$$

Această aproximare permite elaborarea unui model liniar simplificat, dar variabil în timp a sistemului din ecuația (3.16). Această metoda diferă de cea a unei liniarizări obişnuite în jurul unui punct operational fix, prin faptul că traiectoria de referință (care trebuie cunoscuta) este baza liniarizarii şi de aceea aproximarea ia în calcul neliniaritatea din orizontul de estimare. Modelul liniar variabil în timp rezultat, este descris de ecuația:

$$x(k+1) = A(k) * x(k) + b(k) * \Delta u(k) \quad k * v(k) \tag{3.18}$$

cu

$$A(k) = A + k * c^T * \frac{dNL}{dy}|_{y=r(k)} \tag{3.19}$$

$$b(k) = b + k * d * \frac{dNL}{dy}|_{y=r(k)} \tag{3.20}$$

$$v(k) = NL(r(k)) - r(k) * \frac{dNL}{dy}|_{y=r(k)} \tag{3.21}$$

Ecuația de ieşire a sistemului [1], nu este afectată de liniarizare. Când conducem sistemul de-a lungul traiectoriei de referință, neliniaritatea are acelaşi efect ca şi un semnal perturbator variabil in timp v(k). Ca urmare, controlerul ce urmează a fi dezvoltat va trebui să conducă sistemul aproape de traiectoria de referință şi să nu permită abateri mari. Estimarea stărilor viitorului sistem şi a semnalului de ieşire este perfecționată cu modelul din ecuația (3.18), unde valorile A(k), b(k) şi v(k) sunt toate cunoscute în cadrul orizontului prezis; ele depind numai de traiectoria de referință r(k).

3.3.2 Legile controlului predictiv

În această secțiune, introducem legile controlului predictiv pentru cazurile fără restricții şi cu restricții. Indicele de performanță al controlerului de estimare este o sumă a pătratelor erorilor de control.

$$J = \sum_{j=N_1}^{N_2} (r(k+j) - \tilde{y}(k+j))^2 + \lambda \cdot \sum_{j=0}^{N_u} (\Delta u(k+j))^2 \tag{3.22}$$

Variabilele marcate cu ~ sunt valori estimate. În ecuația (3.22), k este momentul de timp curent. Orizontul superior N_2 ar trebui ales astfel încât răspunsurile timpului dominant să fie cuprinse în acest orizont. Cu orizontul inferior N_1 este posibil să apară erori de control la începutul orizontului şi acestea să fie eliminate, numai între N_1 şi N_2. Orizontul de control N_u indică numărul paşilor de control permis în cadrul orizontului. După N_u eşantioane de control, intrarea sistemului Δu este zero(u este constant). Aceasta este o măsură uzuală pentru a reduce cheltuielile de calcul, cu toate că numai o soluție suboptimală este găsită [1,8]. Factorul de evaluare λ ajustează relația dintre evaluarea erorilor de control şi mişcările de control.

Expresiile semnalelor de ieşire, pentru trei eşantioane de control sunt :

$$\begin{aligned}\tilde{y}(k+1) &= \mathbf{c}^T \cdot (\mathbf{A}(k) \cdot \mathbf{x}(k) + \mathbf{b}(k) \cdot \Delta u(k) + \\ &+ \mathbf{k} \cdot v(k)) + d\Delta u(k+1) + \end{aligned} \tag{3.23}$$

$$\begin{aligned}\tilde{y}(k+2) &= \mathbf{c}^T (\mathbf{A}(k+1)\mathbf{A}(k)\mathbf{x}(k) + \\ &+ \mathbf{A}(k+1)\mathbf{b}(k)\Delta u(k) + \mathbf{b}(k+1)\Delta u(k+1) + \\ &+ \mathbf{A}(k+1)\mathbf{k}v(k) + \mathbf{k}v(k+1)) + d\Delta u(k+2) + \end{aligned} \tag{3.24}$$

$$\begin{aligned}\tilde{y}(k+3) &= \mathbf{c}^T (\mathbf{A}(k+2)\mathbf{A}(k+1)\mathbf{A}(k)\mathbf{x}(k) + \\ &+ \mathbf{A}(k+2)\mathbf{A}(k+1)\mathbf{b}(k)\Delta u(k) + \\ &+ \mathbf{A}(k+2)\mathbf{b}(k+1)\Delta u(k+1) + \\ &+ \mathbf{b}(k+2)\Delta u(k+2) + \\ &+ \mathbf{A}(k+2)\mathbf{A}(k+1)\mathbf{k}v(k) + \\ &+ \mathbf{A}(k+2)\mathbf{k}v(k+1) + \mathbf{k}v(k+2)) + \\ &+ d\Delta u(k+3) + \end{aligned} \tag{3.25}$$

Această schemă poate fi continuată până la orizontul superior de estimare N_2. Pentru a facilita notațiile, este convenabil să definim următorii vectori care conțin semnale:

$$\tilde{\mathbf{y}} = \begin{bmatrix} y(k+N_1) & \dots & y(k+N_2) \end{bmatrix} \tag{3.26}$$

$$\Delta\mathbf{u} = \begin{bmatrix} \Delta u(k) & \dots & \Delta u(k+N_u) \end{bmatrix} \tag{3.27}$$

$$\mathbf{v} = \begin{bmatrix} v(k) & \dots & v(k+N_2-1) \end{bmatrix} \tag{3.28}$$

$$\mathbf{r} = \begin{bmatrix} r(k+N_1) & \dots & r(k+N_2) \end{bmatrix} \tag{3.29}$$

unde v si r sunt cunoscuți aprioric, din moment ce depind numai de traiectoria de referință. Semnalele de ieşire estimate ỹ sunt exprimate sub forma unor matrici-vectori de forma:

$$\mathbf{y} = \mathbf{F} \cdot \mathbf{x}(k) + \mathbf{H} \cdot \Delta\mathbf{u} + \mathbf{G} \cdot \mathbf{v} \tag{3.30}$$

Matricile F, H şi G sunt derivate din ecuațiile (3.23), (3.24), (3.25). Ele sunt generate după următoarele reguli:

$$\mathbf{F} = \begin{bmatrix} \mathbf{c}^T \prod_{n=N_1-1}^{0} \mathbf{A}(k+n) \\ \mathbf{c}^T \prod_{n=N_1}^{0} \mathbf{A}(k+n) \\ \vdots \\ \mathbf{c}^T \prod_{n=N_2-1}^{0} \mathbf{A}(k+n) \end{bmatrix} \tag{3.31}$$

$$\mathbf{H} = \begin{bmatrix} h_{N_1-1,N_1-1} & \cdots & h_{N_1-1,N_1-N_u} \\ \vdots & \ddots & \vdots \\ h_{N_2-1,N_2-1} & \cdots & h_{N_2-1,N_2-N_u} \end{bmatrix} \tag{3.32}$$

Elementele matricei H sunt :

$$h_{i,j} = \begin{cases} \mathbf{c}^T \left[\prod_{n=i}^{i-j+1} \mathbf{A}(k+n) \right] \mathbf{b}(k+i-j) & : j > 0 \\ \mathbf{c}^T \mathbf{b}(k+i-j) & : j = 0 \\ d & : j = -1 \\ 0 & : j < -1 \end{cases} \tag{3.33}$$

Matricea G contine efectele neliniarității şi este definită de :

$$\mathbf{G} = \begin{bmatrix} g_{N_1-1,N_1-1} & \cdots & g_{N_1-1,N_1-N_2} \\ \vdots & \ddots & \vdots \\ g_{N_2-1,N_2-1} & \cdots & g_{N_2-1,N_2-N_2} \end{bmatrix} \tag{3.34}$$

Dimensiunile lui H si G sunt diferite, din moment ce orizontul de control N_u are efect numai asupra lui H. Elementele lui G sunt :

$$g_{i,j} = \begin{cases} \mathbf{c}^T \left[\prod_{n=i}^{i-j+1} \mathbf{A}(k+n) \right] \mathbf{k} & : j > 0 \\ \mathbf{c}^T \mathbf{k} & : j = 0 \\ 0 & : j < 0 \end{cases} \tag{3.35}$$

Cu ordinul N al sistemului din ecuația (3.18), matricile au următoarele dimensiuni:

$$\mathbf{F} \in \mathbb{R}^{N_2-N_1+1\times N} \quad \mathbf{H} \in \mathbb{R}^{N_2-N_1+1\times N_u} \tag{3.36}$$

$$\mathbf{G} \in \mathbb{R}^{N_2-N_1+1\times N_2} \tag{3.37}$$

Valoarea funcției din ecuația (3.22) este acum rescrisă în notația matricei vector şi problema de minimizare este soluționată. Utilizând notațiile din ecuatiile (3.30) … (3.34), valoarea funcției devine:

$$J = (\mathbf{r} - \mathbf{F}\mathbf{x}(k) - \mathbf{H}\Delta\mathbf{u} - \mathbf{G}\mathbf{v})^T (\mathbf{r} - \mathbf{F}\mathbf{x}(k) - \mathbf{H}\Delta\mathbf{u} - \mathbf{G}\mathbf{v}) + \lambda\Delta\mathbf{u}^T\Delta\mathbf{u} \tag{3.38}$$

Soluția pentru minimizarea ecuației (3.38) ne dă vectorul acțiunilor de control Δu . Primul element din Δu este folosit ca şi semnal de intrare pentru procesul real, toate celelalte elemente nu sunt folosite pentru control, dar pot servi ca şi valori inițiale pentru următoarea optimizare. Minimizarea ecuației (3.38) şi calculul matricilor necesare, sunt repetate la fiecare pas de integrare. Procedura de minimizare depinde de luarea în considerare sau nu a condițiilor inițiale.

3.3.3 Minimizări fără condiții

În absența condițiilor, minimul lui J poate fi calculat analitic. Egalând gradientul lui J cu zero :

$$\frac{\partial J(\Delta u)}{\partial \Delta u} = \mathbf{0} \tag{3.39}$$

şi rezolvând ecuația liniară rezultantă, soluția optimală pentru Δu este :

$$\Delta\mathbf{u} = \left(\mathbf{H}^T\mathbf{H} + \lambda\mathbf{I}\right)^{-1}\mathbf{H}^T\left(\mathbf{r} - \mathbf{F}\mathbf{x}(k) - \mathbf{G}\mathbf{v}\right) \tag{3.40}$$

Poate fi demonstrat cu uşurință, că matricea $\mathbf{H}^T\mathbf{H} + \lambda\mathbf{I}$(care este termenul pătratic al ecuației (3.38)) este simetrică şi pozitiv definită pentru orice λ pozitiv, ceea ce demonstrează că inversul există întotdeauna şi optim este un minim unic. Primul element al Δu este folosit ca şi semnal de intrare pentru proces. Următorii paşi trebuiesc repetați la fiecare etapă de integrare : calculul matricilor F, H şi G, minimizarea prin evaluarea ecuației (3.40) şi extragerea primului element al Δu. Complexitatea calculelor comparată cu sistemele liniare invariante în timp, este crescută numai datorită recalcularii matricilor F şi G datorită variației de timp a parametrilor modelului predictiv.

Parametrii de control N_1, N_2, N_u şi λ trebuie ajustati în conformitate cu constantele timpului principal ale proceselor şi în funcție de viteza dorită pentru buclele dinamice închise. Stabilitatea nu este garantată pentru orice valori ale acestor parametri[11]. Pentru analiza stabilității, indiferent de valoarea parametrilor controlerului, poate fi adoptat conceptul controlului predictiv al orizontului infinit.

3.3.4 Minimizări cu condiții

Dacă considerăm condiții inițiale semnalelor de control : $\Delta u(k)$, u(k) şi stările x(k), valoarea funcției (3.38) rămâne aceeaşi. Această secțiune se ocupă numai de condiții inițiale impuse semnalului de intrare u şi Δu, dar conditiile de stare pot fi luate in calcul intr-un mod similar. Condițiile inițiale semnalelor de intrare sunt împărțite în două tipuri de inegalități : una pentru condiții asupra incrementării controlului ale lui Δu şi una pentru semnalul de intrare rezultat u. Incrementările controlului nu pot depăşi o anumită valoare minimă şi maximă, după cum este definit în relația (3.41).

$$\Delta u_{min} \leq \Delta u(k+j) \leq \Delta u_{max} \quad \forall \; j = 0 \ldots N_u \tag{3.41}$$

Ecuația (3.41) trebuie să fie validă pentru toate momentele de timp din interiorul orizontului de control. Limitele asupra Δu pot fi combinate în următoarea inegalitate liniară cu $N_{\Delta u}$ =I :

$$\mathbf{N}_{\Delta u} \cdot \Delta \mathbf{u} \leq \mathbf{g}^{u}_{\Delta u} \tag{3.42}$$

$$-\mathbf{N}_{\Delta u} \cdot \Delta \mathbf{u} \leq \mathbf{g}^{l}_{\Delta u} \tag{3.43}$$

Vectorii $g^{u}_{\Delta u}$ si $g^{i}_{\Delta u}$ sunt definiți astfel :

$$\mathbf{g}^{u}_{\Delta u} = [\Delta u_{max} \ldots \Delta u_{max}]^T \tag{3.44}$$

$$\mathbf{g}^{i}_{\Delta u} = [-\Delta u_{min} \ldots -\Delta u_{min}]^T \tag{3.45}$$

În fiecare aplicatie practica ,semnalul de intrare u este de asemenea limitat datorita saturarii actuatorului.Aceste tip de conditii este dat de :

$$u_{min} \leq u(k+j) \leq u_{max} \quad \forall \; j = 0 \ldots N_u \tag{3.46}$$

Din moment ce termenul liber al funcției (3.38) este Δu, ecuația (3.46) trebuie transformată într-o egalitate liniară în Δu. Semnalul de intrare u(k+j) poate fi exprimat astfel :

$$u(k+j) = u(k-1) + \sum_{i=0}^{j} \Delta u(k+i) \tag{3.47}$$

unde valoarea u(k-1) este cunoscută la momentul de timp k. Inegalitățile din ecuația (3.46) pot fi rescrise în funcție de variabila optimizată Δu.

$$\sum_{i=0}^{j} \Delta u(k+i) \leq u_{max} - u(k-1) \tag{3.48}$$

$$-\sum_{i=0}^{j} \Delta u(k+i) \leq -u_{min} - u(k-1) \tag{3.49}$$

Ecuațiile (3.48) şi (3.49) trebuie introduse în orizontul de control pentru j=0….N_u şi sunt din nou transformate într-un sistem liniar de inecuatii :

$$N_u \cdot \Delta u \leq g_u^u \tag{3.50}$$

$$-N_u \cdot \Delta u \leq g_u^l \tag{3.51}$$

Am utilizat următoarele definiții :

$$\mathbf{N_u} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 1 & 1 & & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 1 & 1 & \cdots & 1 \end{bmatrix} \tag{3.52}$$

$$\mathbf{g_u^u} = [u_{max} - u(k-1) \ldots u_{max} - u(k-1)]^T \tag{3.53}$$

$$\mathbf{g_u^l} = [-u_{min} + u(k-1) \ldots - u_{min} + u(k-1)]^T \tag{3.54}$$

Inegalitățile (3.42), (3.43), (3.50) şi (3.51) pot fi combinate într-o singură inegalitate de condiții, astfel că următorul program patratic :

$$\min_{\Delta u} J \quad \text{with} \quad \mathbf{N} \cdot \Delta \mathbf{u} < \mathbf{g} \tag{3.55}$$

N şi g conțin toate matricile N_i , respectiv toți vectorii g_i^j . Programul pătratic din ecuația (3.55) trebuie rezolvat la fiecare pas de integrare cu algoritmi numerici eficienți [13,14].

Liniarizând neliniaritatea izolată de-a lungul traiectoriei de referință , a fost posibil utilizarea unui model predictiv liniar variabil în timp exact şi reducerea problemei de optimizare rezultată în prezența conditiilor inițiale, la un program pătratic. Acest lucru face ca metoda predictivă de control propusă să fie atractivă pentru controlul în timp real al proceselor rapide, unde tehnicile de programare neliniara nu sunt posibile.

3.4 MODELUL CONTROLULUI PREDICTIV DESENSIBILIZAT

În cadrul acestui paragraf vom prezenta o soluție de model de control predictiv, care încorporează sensibilitatea traiectoriei ca să îmbunătățească robustețea strategiei convenționale a MPC. Rezultatele obținute sugerează că abordarea propusă poate reprezenta o alternativă la strategiile robuste propuse pentru modelele de control predictiv, atunci când acestea conduc la cerințe de calcul excesive sau conservatoare.

3.4.1 Analiza controlului predictiv desensibilizat

Considerăm funcționarea controlul predictiv desensibilizat modelat matematic prin ecuația:

$$x(k+1) = A(\mu)x(k) + B(\mu)u(k), x(0) = x_0 \tag{3.56}$$

unde: x reprezintă starea momentană şi u este intrarea de comandă. Matricele spațiului de stare A şi B sunt funcții C^1 ale vectorului parametru $\mu \in R^m$. Parametrul μ , cu valoarea nominală μ_0 , va fi folosit să reprezinte inexactitatea modelului.

În plus, considerăm mărimea de intrare a controlului $u \in U$, unde U reprezintă un set de date politopic şi $0 \in U$.

Fie sensibilitatea traiectoriei p, evaluată la $\mu = \mu_0$:

$$p(k) = \nabla_\mu x|_{\mu=\mu_0} \tag{3.57}$$

Combinând (3.54) cu (3.55), se obține sensibilitatea p a traiectoriei, care poate fi reprezentată cu următoarea ecuație diferențială liniară:

$$p(k+1) = A_\mu x(k) + A_b p(k) + B_\mu u(k) + B_b \mu_\mu(k) \tag{3.58}$$

unde $p(0) = p_0$ şi

$$\begin{cases} A_\mu = \nabla_\mu A(\mu)|_{\mu=\mu_0} \\ B_\mu = \nabla_\mu B(\mu)|_{\mu=\mu_0} \\ \mu_\mu(k) = \nabla_\mu u(k)|_{\mu=\mu_0} \\ A_b = \mathrm{Im} \otimes A(\mu_0) \\ B_b = \mathrm{Im} \otimes B(\mu_0) \end{cases} \tag{3.60}$$

unde: $\otimes$ reprezintă simbolul produsului Kroeneker al celor două matrice şi Im reprezintă identitatea ordinii “m”.

Sensibilitatea traiectoriei furnizează informații despre modul în care parametrii de stare se schimbă in variații infinitezimale ale lui μ, în jurul valorii nominale a lui μ_0 Acestă observație va fi folosită pentru a reduce sensibilitatea predicției stării la valoarea parametrului model în MPC.

MPCD este o tehnică de control în care acțiunea de comandă la fiecare prelevare de probe este obținută prin rezolvarea unui program pătratic on-line, folosind starea curentă a mecanismului ca stare inițială. În această secțiune funcția cost optimizată a lui MPC de la momentul *k* este:

$$J_{MPCD} = \sum_{j=k}^{\infty} \|x(j)\|_Q^2 + \|u(j)\|_R^2 \tag{3.60}$$

unde: Q şi R sunt matrice pozitive de dimensiuni adecvate. În relația (3.60) mărimea de intrare a controlului u(j) este fixată la zero după N trepte de timp, unde N este controlul orizontului. Ca să se reducă sensibilitatea predicției de stare prin alegerea lui μ în model, introducem un termen de cost care penalizează energia traiectoriei de sensibilitate.

Costul ce urmează a fi folosit în strategia modelului controlului predictiv desensibilizat este determinat după cum urmează:

$$J_{MPCD} = J_{MPCD} + \sum_{j=k}^{\infty} \left\| p(j) \right\|_R^2 \tag{3.61}$$

unde: Q_S este o matrice determinată pozitivă de dimensiuni adecvate care evaluează importanța relativă a sensibilității traiectoriei cu privire la costul MPCD.

Folosind relația (3.61), problema de optimizare a MPC la momentul K este definită prin setul de relații:

$$\begin{cases} \min\limits_{u} J_{MPCD} \\ x(j+1) = Ax(j) + Bu(j) \\ p(j+1) = A_\mu x(j) + A_b p(j) + B_\mu u(j) \\ x(k) = x_k^m,\ p(k) = p_k^m \\ u(j) \in U,\ j = k,...,k+N-1 \\ u(j) = 0,\ j \geq k+N \end{cases} \tag{3.62}$$

unde: x_k^m, p_k^m sunt stări ale modelelor la timpul *k*. Se constată că x_k^m este măsurat în timp, iar p_k^m este o variabilă de proiectare.

3.4.2 Stabilitatea nominală

Problema MPCD reprezintă de fapt o problemă a modelului MPC, complementară cu modelul de sensibilitate.

Rezultatele stabilității nominale sunt evaluate în literatura de specialitate de ceva timp. Noi vom utiliza în continuare rezultatele stabilității nominale care provin din particularizarea soluțiilor rezultate în lucrarea [16].

Presupunând $\mu = \mu_0$ vom determina modelul

$$z(k+1) = \overline{A}z(k) + \overline{B}u(k) \tag{3.63}$$

unde: $z(k)\left[x(k)^T \quad p(k)^T\right]^T$ este starea complementară şi matricile A, B sunt de forma:

$$\overline{A}=\begin{bmatrix} A & 0 \\ A_\mu & A_b \end{bmatrix}, \overline{B}=\begin{bmatrix} B \\ B_\mu \end{bmatrix} \tag{3.64}$$

Notăm cu Z_N setul de condiţii iniţiale pentru care există o succesiune de control $u(k)\in U$, pentru k=0,...,N-1 precum şi u(0)=0. În plus notăm cu $X_N(p_0)$ setul de condiţii iniţiale x_0 pentru care: $\left[x_0^T, p_0^T\right]^T \in Z_N$

Pe baza acestor notaţii şi observaţii, putem enunţa următoarele teoreme:

Teorema 3.1 (Mecanism stabil): pentru $\overline{A}$ stabil şi $N\geq 1$, sistemul închis al MPCD-ului restricţionat este un sistem global asimptotic stabil.

Teorema 3.2 (Mecanism instabil) : pentru setul de valori $\left(\overline{A},\overline{B}\right)$ cu sensibilitatea p_0 şi N≥p_0, mai mare sau egal cu numarul valorilor proprii instabile ale lui A, sistemul închis al MPCD-ului restricţionat este asimptotic stabil în $X_N(p_0)$.

3.4.3 Implementarea

Implementarea directă a problemei MPCD-ului nu este în general posibilă, pentru că necesită rezolvarea unei probleme de optimizare în cazul unui orizont infinit.

Din moment ce mărimea de intrare a controlului este zero, pentru $j\geq K+N$ termenii din J_{MPCD} cu $j\geq K+N$ sunt funcţii doar ale mulţimii {z(K+N)}.

În cazul în care A este stabil, suma acestor termeni este o funcţie pătratică de forma z(K+N). De aceea, problema orizontului infinit poate fi transformată într-un orizont finit echivalent prin înlocuirea costului în J_{MPCD} (3.60) cu expresia:

$$J_{DMPC}=\left\|z(k+N)\right\|_{\overline{P}}+\sum_{j=k}^{k+N-1}\left\|z(j)\right\|_Q^2+\left\|u(j)\right\|_R^2 \tag{3.63}$$

unde $\overline{Q}=diag(Q,Q_s)$ si $\overline{P}$ este soluţia ecuaţiei:

$$\overline{P}=\overline{A}^T\overline{PA}+\overline{Q} \tag{3.64}$$

Când A este instabil trebuie să garantăm că stările corespunzătoare valorilor proprii instabile sunt considerate 0 la capătul orizontului de control.

Definim matricile: A_J ,M, M^{-1} prin relațiile :

$$A_J = \begin{bmatrix} A_{JS} & 0 \\ 0 & A_{JU} \end{bmatrix} \tag{3.67}$$

$$M = \begin{bmatrix} M_S & M_U \end{bmatrix} \tag{3.68}$$

$$M^{-1} = \begin{bmatrix} \overline{M}_S \\ \overline{M}_U \end{bmatrix} \tag{3.69}$$

unde A_{JS} conține toate valorile proprii ale lui $\overline{A}$ cu valori cuprinse în intervalul (0, 1).

Se observă că dacă $z(k) \in Z_N$ atunci MPCD are un cost optimal determinat. Aceasta înseamnă că modurile instabile vor fi nule la momentele de timp K+N [16]. Aceasta este:

$$\overline{M}_U z(k+N) = 0 \tag{3.70}$$

Egalitatea (3.70) reprezintă o condiție necesară şi suficientă pentru a asigura costul optimal determinat. Deoarece z(K+N) aparține subspațiului stabil al lui A, urmând acelaşi procedeu, problema MPCD a orizontului infinit poate fi redusă la studiul orizontului finit într-una a orizontului finit prin încorporarea restricției (3.70) şi înlocuind costul stabilită de relatia (3.65), unde $\overline{P}$ este acum obținut prin rezolvarea ecuației:

$$\overline{P} = \overline{M}_S^T P \overline{M}_S \tag{3.71}$$

sau :

$$P = A_J^T P A_{JS} + M_S^T \overline{Q} M_S \tag{3.72}$$

Rrezultă aşadar că problema de optimizare a orizontului infinit, indiferent dacă matricea A este stabilă sau instabilă, se reduce la un program pătratic bidimensional, determinat cu un termen inclus în cost de la care se asteaptă să se îmbunătățească robustețea.

În următoarea secțiune investigăm potențialul formulării DMPC comparând funcționarea sa cu cea a unui MPC convențional, care este proiectat folosind mecanismul nominal, şi cu abordarea MPC robust (RMPC) descrisă la [BS-07d], care nominalizează o limită superioară a lui $z(k) \in Z_N$ pentru toate valorile posibile de nesiguranță.

3.5 CONTROLUL ADAPTIV ROBUST PENTRU SISTEME AUTONOME

Modelul de control adaptiv prezentat, este caracterizat de un sistem discret neliniar variabil în timp prezentând atât erori variabile cât şi invariabile. Pentru toate serverele active pe acelaşi sistem hardware, Controlul Adaptiv Robust estimează secvența de date corespunzătoare unui process de prelucrare specificat din UCP. Presupunând că pe fiecare calculator activează *j* servere virtuale diferite, atunci Sistemul de Calcul Autonom va activa *j* Controlere Adaptive Robuste diferite (vezi Fig. 3.5). Totuşi, această soluție nu este optimală, întrucât implică un consum mare de resurse de calcul doar pentru a controla încărcarea în memorie, când acestea ar putea fi utilizate în procesele active. Prin urmare, se poate considera că, pentru sisteme hardware diferite, va trebui să implementăm o singură formulă de calcul care va fi activată la perioade de timp dependente de dinamica lentă a serverului care le procesează. Această formulare necesită asocierea mai multor variabile vectoriale descries de următorul set de ecuații:

$$\begin{cases} X_1(K) = \left[x_{1_1}(K), x_{1_2}(K), \ldots, x_{1_j}(K)\right]^T \\ X_2(K) = \left[x_{2_1}(K), x_{2_2}(K), \ldots, x_{2_j}(K)\right]^T \\ X_3(K) = \left[x_{3_1}(K), x_{3_2}(K), \ldots, x_{3_j}(K)\right]^T \end{cases} \tag{3.73}$$

$x_{i\,j}(K)$ fiind componentele de stare ale fiecăruia dintre cele *j* procese.

Având in vedere rezultatele prezentate în lucrarea [10], modelul sistemului de calcul autonom se poate definit prin sistemul de relații:

$$\begin{cases} X_1(K+1) = X_2(K) + \theta^T \alpha_1(X_1(K)) + \eta_1(K) \\ X_2(K+1) = X_3(K) + \theta^T \alpha_2(X_1(K), X_2(K)) + \eta_2(K) \\ X_3(K+1) = \theta^T \alpha_3(X_1(K), X_2(K), X_3(K)) + \eta_3(K) + u(K) \end{cases} \tag{3.74}$$

unde K reprezintă rata de eşantionare, iar X(K) = $\left[X_1(K), X_2(K), X_3(K)\right]^T$ este rata sistemului de calcul autonom (timpul de reacție al aplicațiilor).

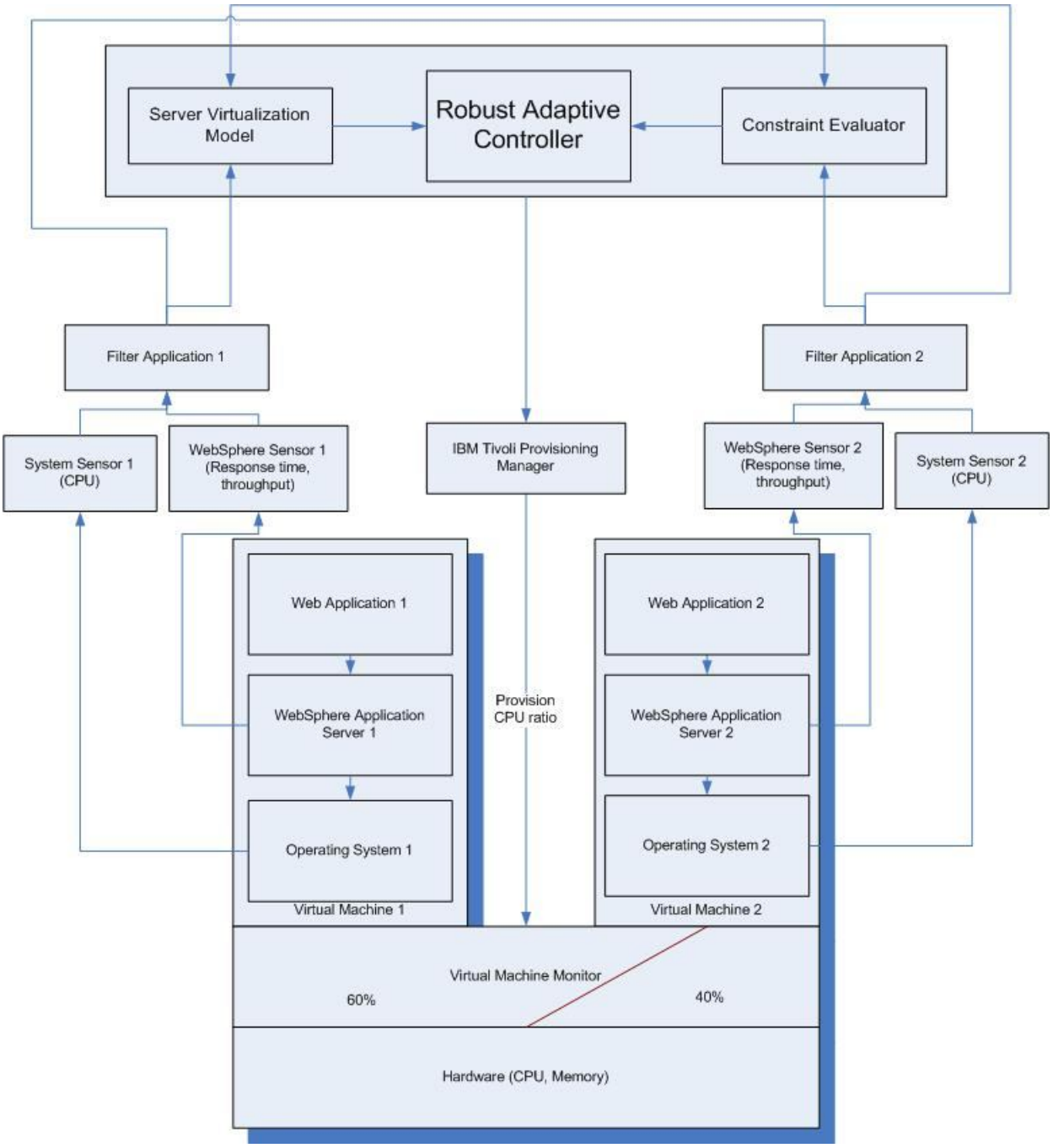


Figura 3.5. Arhitectura unui Sistem de Calcul Autonom Adaptiv Robust pentru controlul incarcării cu ajutorul UCP a unui mediu de server virtual

Cu alte cuvinte, X(K) $\in$ X $\subset R^3$, cât timp θ^T este un vector cu parametrii necunoscuți,

$$\theta^T = [\theta_1, \theta_2, \theta_3]^T; \theta \in \Theta = \begin{Bmatrix} \theta : \|\theta\| \le k_o; \\ \forall \theta \in \Theta \end{Bmatrix} \tag{3.75}$$

α_i sunt funcții cunoscute, obținute, de exemplu, dintr-un process RPE [15].

$$\begin{cases} \alpha_i(X_1, \ldots, X_i(K)) \\ \alpha_i \in C(R^i, R^P) \\ \alpha_i(0) = 0 \\ \eta_i(K) < C_{\eta,i}\varepsilon\delta \\ \alpha_i(K) = \alpha_i(X_1, \ldots, X_i(K)); i = 1,2,\ldots,n \end{cases} \tag{3.76}$$

Presupunem că se verifică următoarea condiție:

$$\eta_i(K) \le C_\eta \varepsilon \left\| [X_1(K), X_2(K), \ldots, X_i(K)] \right\| + d \tag{3.77}$$

unde η_i sunt funcțiile necunoscute datorită dinamicii imprevizibile a procesului de calcul.

În aceste conditii se obține inegalitatea:

$$\left| y(K) - K_y y_m(K) \right| < \varepsilon \tag{3.78}$$

unde y(K) este iesirea sistemului care îndeplineşte condiția : y(K) = X_1(K) şi y_m(t) este un punct de referință arbitrar ales, iar ε este o constantă arbitrar aleasă de valori infime(ε->0).

$$\left\| \alpha_i(X_i(K) - \alpha_i(X_i'(K)) \right\| \le K_\alpha \left\| X_i(K) - X_i'(K) \right\| \tag{3.79}$$

Ecuația de mai sus este îndeplinită de toți termenii $X_i(K), X_i'(K) \in C(R, R^i)$, unde K$\alpha$ este o constanta reală cunoscută.

Legea controlului adaptiv robust, care decide ce parte a UCP se va aloca diferitelor servere virtuale de pe un anumit sistem hardware (calculator) este dată de formula:

$$u(K) = y_m(K+n) - \hat{\theta}^T(K)\alpha_n(K) - \sum_{l=1}^{n-1} \hat{\theta}^T(K)\overline{\alpha}_{l,n}(K) \tag{3.80}$$

Pentru a evita supra-parametrizarea, o binecunoscută deficiență a acestor metode de control adaptiv bazate pe identificator, va trebui să modificăm legea controlului care obligă vectorul Θ (t) să satisfacă egalitatea:

$$\hat{\theta}^T(K+1)=\wp\left\{\hat{\theta}(K)+\frac{\Theta^T(K)\zeta(K+1)}{1+M_0+trace(\Theta(K)\Theta^T(K))}\right\} \tag{3.81}$$

unde:

$$\begin{cases}\zeta(K+1)\triangleq Z(K+1)-F_Z(K)-by_m(K+n)-\Psi(K+1)\\ \phi(K+1)=\left[\eta_1(K),\eta_2(K),\ldots,\eta_n(K)\right]^T\\ \Psi(K+1)=\left[X_1(K+1),X_2(K+1),\ldots,X_n(K+1)\right]^T\end{cases} \tag{3.82}$$

Z(K) e un vector tranformator de stare:

$$Z(K)=\begin{bmatrix}Z_1(K)\\ Z_2(K)\\ Z_3(K)\end{bmatrix} \tag{3.83}$$

ale căror elemente reprezintă soluțiile ecuațiilor:

$$\begin{cases}Z_1(K)=X_1(K)\\ Z_2(K)=X_2(K)+\hat{\theta}^T(K)\alpha_1(K)\\ Z_3(K)=X_3(K)+\hat{\theta}^T(K)(\alpha_1(K)+\alpha_2(K))\end{cases} \tag{3.84}$$

Parametrii α_i (K) satisfac următoarele ecuații:

$$\begin{aligned}\alpha_{l,j}(K)=\alpha_i(&Z_{j-i+1}(K),Z_{j-i+2}(K)-\theta^T\overline{\alpha}_{1,j-i+1}(K),Z_{j-i+m}(K)-\theta^T(K)\sum_{p=1}^{m-1}\alpha_{p,j-i+m}(K),\ldots\\ &\ldots,Z_j(K)-\theta^T(K)\sum_{p=1}^{i-1}\alpha_{p,i-1}(K))\end{aligned} \tag{3.85}$$

În relația de mai sus care permite determinarea lui *u(K)*, din legea controlului adaptiv, variabilele de stare X_i sunt date de relațiile:

$$\begin{cases}X_1(K+1)=0\\ X_2(K+1)\triangleq\sum_{n-1}\theta^T(K+1)\alpha_1(K+1)-\hat{\theta}^T(K)\alpha_1(K)\\ X_n(K+1)\triangleq\sum_{l=1}^{n-1}(\theta^T(K+1)\alpha_{l,n-1}(K+1)-\hat{\theta}^T(K)\alpha_{l,n}(K))\end{cases} \tag{3.86}$$

unde matricea F este dată de expresia:

$$F = \begin{bmatrix} 0 & l_{(n-1)\times(n-1)} \\ 0 & 0 \end{bmatrix} \in \Re^{n\times n} \tag{3.87}$$

Cu această dezvoltare matematică, Controlul Adaptiv Robust pentru sistemele autonome a fost proiectat complet.

Funcțiile α_i (K) sunt obținute ca rezultat al aplicării procedurilor de indentificare online urmând algoritmii de identificare a sistemului [15] cunoscuți aprioric. Calcularea parametrilor sistemului este repetată prin perioade de timp successive, depinzând de variația în timp atât a procesului in sine cât şi a funcțiilor α_i (K). Având în vedere faptul că subiectul construirii unor α_i (K) adecvate este destul de complex, va fi discutat într-o anexă referitoare la aspectele de control în modelarea, auto-asigurarea şi auto-optimizarea sistemelor autonome de calcul.

Pe baza rezultatelor stabilității obținute în lucrarea [16], legea controlului garantează stabilitatea sistemului, cunoscând că inegalitățiile presupuse mai sus sunt adevarate, şi că ||z(t)|| este limitat de toate condițiile inițiale, puncte fixe, şi perturbații. Se presupune de asemenea că eroarea satisface inegalitatea:

$$\sum_{\tau=t_{0+1}}^{t} \left| y(\tau) - y_m(\tau) \right| \le \beta_1 + \beta_2 0(\varepsilon,\delta)(t - t_0) \tag{3.88}$$

unde $\beta 1$ şi $\beta 2$ sunt constante, şi $0(\varepsilon)$ e o funcție care îndeplineşte condiția $\lim_{\substack{\varepsilon\to 0 \\ \delta\to 0}} 0(\varepsilon,\delta) = 0$

Celelalte componente prezente în arhitectura unui sistem de control adaptive în timp real, pentru un mediu autonom de calcul cum ar fi: senzorii, estimatoarele, filtrele, coordonatoarele, componentele cu putere de decizie şi actuatoarele sunt prezentate în lucrarea [17], care conține şi detalii despre condiționarea semnalelor, aproximarea variabilelor de stare sau despre filtarea variabilelor de stare estimate.

3.6 CONCLUZII ŞI CONTRIBUȚII

În cadrul acestui capitol al Tezei de doctorat, am abordat analiza matematică a modelelor predictive de control adaptiv, utilizate în implementarea sistemelor autonome de calcul. Sub denumirea generică „Model predictiv de control" se înțelege de fapt o multitudine de metode, modele şi algoritmi de control al calculatorului, menite să estimeze comportamentul şi răspunsul unui echipament numeric de calcul, la stimului cu variație predictivă.

Au fost prezentate, analizate şi dezbătute următoarele subiecte:

- Strategia generală a modelului predictiv de control pornind de la diferite modele şi funcții de performanță prezentată în literatura de specialitate.
- Analiza modelului Wiener standard, care constituie cel mai reprezentativ model neliniar predictiv de control. În aplicațiile industriale, implementarea acestui model soluționează problemele de control al regulatoarelor şi servomecanismelor supuse la perturbații aleatoare cu o densitate spectrală distribuită într-un domeniu larg al frecvențelor.
- Abordarea unui model predictiv de control neliniar în spațiul multidimensional. Reprezentările şi analiza aplicatiilor industriale ca reprezentând procese liniare invariante în timp, cu condiții initiale nule este o ipoteză simplificatoare care oferă o soluție analitică simplă la costuri scăzute(valabilă în general pentru analiza proceselor industriale lent variabile în timp). În realitate, evolutia rapidă a parametrilor dinamici conduce la modele neliniare cu costuri de calcul sporite în vederea minimelor locale şi globale.
- Au fost introduse legile controlului predictiv ca intrument deestimare a erorilor şi de evaluare a costurilor solicitate de aplicarea modelelor predictive de control adaptiv pentru sistemele autonome de calcul.
- Analiza modelului de control predictiv desensibilizat a demonstrat că reprezintă o alternativă viabilă la strategiile robuste propune în literatura de specialitate pentru modelele de control predictiv, atunci când acestea conduc la calcule excesive.
- Controlul adaptiv robust estimează secvența de date corespunzătoare unui proces de prelucrare specificat al UCP. Această abordare nu este optimală deoarece implică un consum mare de resurse de calcul, numai pentru a controla încărcarea datelor în memorie şi nu pot fi utilizate în procesele de prelucrare interactivă.

Principalele contribuții originale aduse la elaborarea acestui capitol au fost publicate de autor în reviste de specialitate şi pot fi sistematizate astfel:

- Identificarea şi stabilirea nucleului modelelor predictive de control adaptival sistemelor autonome de calcul.

- Descrierea şi analiza modelului generic de control adaptiv, cu identificarea avantajelor şi dezavantajelor pe care le implică utilizarea modelelor standard.

- Se demonstrează că, în prezent, tehnologiile comerciale ale MPC disponibile, au la bază implementarea unui model liniar de control al sistemului. Procesele industriale reale cu variație rapidă în timp a parametrilor prezintă o neliniaritate a funcției de transfer ce nu poate fi ignorată. Analiza pe acească bază, conduce la scăderea performanțelor globale, datorită incapacității modelelor liniare de a aproxima cu precizie procesele neliniare.

- Analiza atentă a modelului Wiener standard, care este un model neliniar predictiv de control, a permis încorporarea sa într-o schemă activă de prelucrare în care neliniaritatea a fost eficient aproximată cu o schemă de control predictiv liniar.

- A fost elaborată o schemă originală de model predictiv de control (Fig. 3.3) alcătuită din două subsisteme în caresemnalele la ieşirea blocului de prelucrare a semnalelor neliniare se aplică la intrarea unui bloc de inversare care anulează neliniaritatea.

- Solutia analitică dezvoltată în cadrul paragrafului 3.2 simplifică problema optimizării, care, pentru această situație se reduce la analiza unei probleme de programare pătratică standard.

- Schema propusă pentru WMPC permite utilizarea unui program pătratic de calcul(QP) convex, care se poate rezolva eficient utilizând algoritmii de calcul existenți.

- A fost efectuat un studiu al modelelor predictive de control neliniar în spațiul multidimensional bazate pe analiza în timp real a funcției de transfer pentru sistemele liniare discrete.

- S-a introdus conceptul de model predictiv de control cu neliniaritate izolata, care combină avantajele analizei proceselor de predicție liniară şi neliniară.

- În ipotezele de calcul specificate, problema optimizării sistemelor cu neliniaritate izolată poate fi rezolvată analitic şi nu necesită utilizarea algoritmilor numerici de optimizare care cresc costurile sistemelor de prelucrare.

- Stabilirea condițiilor de minimizare a erorilor de prelucrare pentru cele două corpuri limită: cu şi fără conditii inițiale nule.

- Elaborarea unui model de control predictiv desensibilizat ca alternativă viabilă la strategiile robuste propuse în literatura de specialitate pentru modelele predictive de control adaptiv.

- Se demonstrează că strategia MPCD reprezintă o tehnică modernă de control adaptiv în care acțiunea de comandă poate fi soluționată prin rezolvarea on line a unei ecuații de gradul doi, folosind starea curentă a mecanismului ca stare inițială.

- Elaborarea a două teoreme (mecanism stabil şi instabil) care stau la baza analizei stabilității nominale a sistemului proiectat.

- Analiza sistemelor de control adaptiv robust caracterizate prin semnale de intrare afectate de erori variabile şi invariabile. Se demonstrează că această soluție nu este optimală, deoarece implică un consum mare de resurse de calcul.

- Pentru a evita supra-parametrizarea modelelor de control adaptiv bazate pe identificator, s-a propus o constrângere suplimentară (relația 3.81), care trebuie aplicată vectorului ce descrie legea controlului.

- Pe baza analizei efectuate asupra stabilității sistemelor de control autonom adaptiv s-a dedus empiric o condiție de inegalitate (relația 3.88) pe care o satisfac erorile produse la prelucrare.

Studiul inițiat în cadrul acestui capitol a demonstrat că există o multitudine de modele adaptive pentru controlul predictiv al sistemelor autonome de calcul, bazate pe modele de stări în spațiul neliniar. Pentru o analiză facilă este necesar să se facă o separare între partea de prelucrare neliniară şi răspunsul sistemului, care este de obicei neliniar. Majoritatea algoritmilor propuşi pentru modelele predictive de control a sistemelor autonome neliniare necesită soluționarea unei probleme de optimizare nonconvexe şi neliniare pe durata pasului de eşantionare. Acest lucru este practic imposibil în sistemele reale care au o rată de eşantionare mare. Pentru a facilita aplicarea algoritmului predictiv de control pentru aplicații în timp real, lucrarea propune o metodă de liniarizare pe porțiuni a răspunsului neliniar al sistemului, pe durata traiectoriei de referință. Ca urmare a rezultat un medel liniar variabil în timp, in care neliniaritatea răspunsului va fi descrisă de variația temporară a modelului. Pasul de predicție şi legea de control predictiv pentru un sistem dat pot fi calculate analitic cu relațiile deduse în lucrare, în condiții inițiale specificate.

4. ALGORITMI DE PREDICTIE PENTRU ESTIMAREA ERORILOR IN SISTEME DIGITALE

Î n contextul dezvoltarii tehnologice actuale, infrastructura informationala inglobeaza aplicatii diverse,care conduc la sisteme nesigure,dificil de gestionat.Prezenta semnalelor aleatoare(zgomote) la intrarile sistemelor digitale de calcul,necesita utilizarea modelelor de control predictiv[6],pentru analiza si controlul parametrilor de proces.Lucrarea initiaza un studiu privind implementarea algoritmilor de predictie in arhitectura sistemelor autonome de calcul si evalueaza erorile de modelare produse la predictie.Aplicarea formalismului matematic dezvoltat,la studiul comportarii unui sistem digital de control adaptiv dinamic, este benefica la prelucrarea si controlul unei cantitati minime de informatii.

4.1 PREDICȚIA PROCESELOR ALEATOARE

Considerăm un sistem digital asupra căruia acționează simultan semnale aleatoare şi deterministe. Dacă observațiile asupra sistemului se efectuează pe durata unui interval de timp finit, semnalul nul aplicat la intrarea sistemului u(t) poate fi exprimat analitic cu relația:

$$u(t) = f(t) + s(t) + n(t) = x(t) + n(t) \tag{4.1}$$

unde : x(t) reprezintă semnalul util aplicat la intrare, f(t) – partea sistematică din semnalul util, s(t) – partea aleatoare, iar n(t) – zgomotul care se suprapune la intrare.

Partea sistematică a semnalului este:

$$f(t) = \sum_{k=1}^{m} c_k \cdot f_k(t) \tag{4.2}$$

reprezintă o combinație liniară de funcții cunoscute f_k, $k \in \{1, \ldots, m\}$, iar c_k sunt constante aleatoare necunoscute.

Procesele aleatoare s(t) şi n(t) staționare şi necorelate pot fi reprezentate prin următoarele funcții de corelație:

$$\begin{cases} R_s(t_1, t_2) = M[s(t_1), s(t_2)] \\ R_n(t_1, t_2) = M[n(t_1), n(t_2)] \end{cases} \quad (4.3)$$

unde M reprezintă operatorul de mediere.

Dacă transformarea efectuată asupramsemalului de intrare x(t) este liniară, atunci semnalul la ieşirea din sistem poate fi calculat cu relația:

$$y(t) = \int_0^{\infty} g(t, \xi) \cdot x(\xi) d\xi \quad (4.4)$$

unde: g(t, ξ) reprezintă funcția de pondere.

Metoda prin care procesul aleator: $\xi(t)$, $t = \{t \in T; t < t_0\}$ poate fi estimat în punctul t* = t_0+τ, τ >0 se numeşte predictie (o particularizare a noțiunii de extrapolare).

Aşadar predicția realizează estimarea unui proces aleator în punctele: t*=t_0 în care se efectuează măsurarea semnalelor de intrare sau de ieşire.

În dezvoltările ulterioare vom considera că semnalul util x(t) şi zgomotul n(t) sunt semnale aleatoare neconstante în care predicția delimitează semnalul util de zgomot.

4.1.1 Estimarea predicției proceselor aleatoare

Considerăm că valoarea semnalului de intrare x(t_2) urmează să fie estimată pe baza semnalului de ieşire y(t_1) măsurat la un moment de timp anterior t_1<t_2. Predicția procesului aleator poate fi modelată cu ajutorul relației:

$$x(t_2) = \phi(t_2, t) \cdot x(t) + \int_{t_1}^{t_2} \phi(t_2, \tau) \cdot B(\tau) \cdot W(\tau) d\tau \quad (4.5)$$

unde: $x(t_2)$ reprezintă valoarea semnalului de intrare obținută prin măsurători la momentul de timp t_1, $\phi(t, \tau)$ matricea de transfer a procesului, w(t) – funcția de distribuție a procesului aleator sub forma unui zgomot alb, iar B(t) o matrice de corecție constantă. Dacă modelul de predicție analizat nu este afectat de perturbații (zgomote) va fi satisfăcută condiția [3]

$$\frac{\partial}{\partial t}[\phi(t, \tau)] = \mathbf{A}(t) \cdot \phi(t, \tau) \cdot \phi(\tau, t) = I \quad (4.6)$$

unde: A(t) este o constantă care se poate determina din condițiile inițiale, iar I reprezintă operatorul unitar.

Calculăm derivata produsului:

$$\frac{\partial}{\partial \tau}[\phi(t,\tau)\cdot\phi(\tau,t)] = \phi(\tau,t)\left[\frac{\partial}{\partial \tau}\phi(t,\tau)\right] + \phi(t,\tau)\left[\frac{\partial}{\partial \tau}\phi(\tau,t)\right] = 0 \tag{4.7}$$

Comparând ultimele două relații rezultă:

$$\phi(\tau,t)\left[\frac{\partial}{\partial \tau}\phi(t,\tau)\right] = -\phi(t,\tau)\cdot A(\tau)\cdot\phi(\tau,t) \tag{4.8}$$

sau:

$$\frac{\partial}{\partial \tau}\phi(t,\tau) = -\phi(\tau,t)\cdot A(\tau) \tag{4.9}$$

Pentru a obține informații suplimentare vom aplica relației (4.5) operatorul de mediere: M. Se obține:

$$\hat{x}(t_2|t_1) = \phi(t_2,t_1)\cdot\hat{x}(t_1) + \int_{t_1}^{t_2}\phi(t_2,\tau)\cdot B(\tau)\cdot M\{w(\tau)|y(t_1)\}d\tau \tag{4.10}$$

unde, am notat $\hat{x}(t_2|t_1) = M\{x(t_2)|y(t_1)\}, \hat{x}(t_1) = M\{x(t_1)|y(t_1)\}$

Deoarece am presupus procesul aleator staționar sub forma unui zgomot alb, este îndeplinită condiția[4]:

$$M\{w(\tau)|y(t_1)\} = M\{w(\tau)\} = 0 \text{ , pentru } \tau \geq t_1 \tag{4.11}$$

Utilizând această condiție în relația (4.10) se obține:

$$\hat{x}(t_2|t_1) = \phi(t_2,t_1)\cdot\hat{x}(t_1) \tag{4.12}$$

Această relație reprezintă suportul matematic al algoritmului de predicție, valabil atât pentru estimarea semnalelor continuale (analogice) cât şi discrete (digitale). În funcție de natura variației timpilor: t_1 şi t_2 se identifică trei tipuri de predicție:

a) Predicția pe interval fix: t_1= constant şi t_2=t variabil

b) Predicția cu punct fix: t_2= constant şi t_1=t variabil

c) Predicția cu avertizare în care: t_1=t şi t_2=t+T, unde T reprezintă perioada de eşantionare.

Vom prezenta expresiile analitice care modelează cele trei tipuri de predicție definite anterior.

PIF Pentru predicția pe interval fix, relația (4.12) devine:

$$\hat{x}(t_2|t_1) = \phi(t_2,t_1)\cdot\hat{x}(t_1), t > t_1 \tag{4.13}$$

Estimația semnalului de intrare $\hat{x}(t_1)$ se poate calcula utilizând funcția de transfer a unui filtru Kalman-Bucy [5]. Elementele matricii funcției de transfer a sistemului se obțin prin calcule, rezolvând ecuația diferențială:

$$\frac{\partial}{\partial \tau}\phi(t,t_1) = A(t)\cdot\phi(t,t_1)$$

cu condițiile inițiale, specifice aplicației analizate $(\phi(t,t_1) = I)$

PPF Predicția cu punct fix, realizează estimația stării sistemului într-un moment fix viitor, utilizând stările sistemului la un moment de timp anterior (care poate fi variabil).

În această situație se poate scrie:

$$\hat{x}(t_2|t) = \phi(t_2,t)\cdot\hat{x}(t_1), t \le t_2 \qquad (4.14)$$

Deoarece, în această situație, cel de-al doilea argument al matricii funcției de transfer este variabil, vom scrie ecuația (4.9) sub forma:

$$\frac{\partial}{\partial \tau}\phi(t_2,t) = -\phi(t_2,t)\cdot A(t)$$

unde: $\phi(t_1,t_2) = I$.

PA Predicția cu avertizare realizează estimarea stării sistemului la momentul de timp T în raport cu momentul trecut t. Algoritmul predicției poate fi scris acum sub forma:

$$\hat{x}(t+T|t) = \phi(t+T,t)\cdot\hat{x}(t) \qquad (4.15)$$

Deoarece, în acest caz, ambele argumente ale funcției de transfer sunt variabile, ecuația (4.9) poate fi scrisă:

(4.16)

$$\frac{d}{dt}[\phi(t+T,t)] = \left[\frac{\partial}{\partial\tau}\phi(\tau,t)\right]\Big|_{\tau=t+T} + \left[\frac{\partial}{\partial\tau}\phi(\tau,t)\right]\Big|_{\tau=t+T}$$

în care:

$$\frac{\partial}{\partial t}[\phi(t+T,\tau)\cdot\hat{x}(t)] = A(t+T)\cdot\phi(t+T,t) - \phi(t+T,t)\cdot A(t)$$

Această ecuație diferențială poate fi rezolvată cu condiția la limită $\phi(t_0+T,t_0) = \phi_0$ care se obține ca soluție a ecuației (4.6) prin integrare între limitele t_0 şi t_0+T.

Se constată că cel mai simplu caz îl reprezintă predicția pe interval fix, în care se estimează o stare variabilă pe baza unei mulțimi de rezultate obținute prin măsurări asupra semnalului de intrare la momentul de timp fix t_1.

4.1.2 Analiza erorilor produse la predicție

Estimarea stării viitoare a unui sistem digital poate fi corespunzătoare dacă erorile de modelare se încadrează in limite impuse, intervalul de estimare este bine definit şi dispersia zgomotului prezent la

intrare se situează într-o bandă limitată. Din acest motiv este necesar să evaluăm şi să controlăm dispersia erorilor de predicție. Eroarea absolută de predicție se calculează cu relația:

$$\hat{\varepsilon}_x(t_2, t_1) = |x(t_2) - \hat{x}(t_2|t_1)| \tag{4.16}$$

şi similar, eroarea relativă de predicție

$$\hat{\varepsilon}_r(t_2, t_1)(\%) = \frac{|x(t_2) - \hat{x}(t_2|t_1)|}{\hat{x}(t_2|t_1)} \cdot 100 \tag{4.17}$$

Utilizând relațiile (4.5) şi (4.12) vom putea scrie:

$$[\varepsilon_1 x)\hat{}(t_1 2 \mid t_1 1) = \phi(t_1 2, t_1 1) \cdot (\varepsilon_1 x)\hat{}(t_1 1) + \int_1(t_1 1)^{\intercal}(t_1 2) \cdot \int_{t1}^{t2} [\phi(t_1 2, \tau) \cdot B(\tau) \cdot w(\tau) d\tau] \tag{4.18}$$

Considerând că erorile $\hat{\varepsilon}_x(t_1)$ şi $w(\tau)$ sunt mărimi necorelate pentru $\tau \geq t_1$, se poate obține expresia dipsersiei erorii de estimare a procesului de predicție:

$$\left[\sigma_\downarrow \varepsilon^{\uparrow} 2\right)\hat{}(t_\downarrow 2 \mid t_\downarrow 1) = D\left[(\varepsilon_\downarrow x)\hat{}(t_\downarrow 2 \mid x_\downarrow 1)\right] = o(t_\downarrow 2, t_\downarrow 1) \cdot \left(\sigma_\downarrow x^{\uparrow} 2\right)\hat{}(t_\downarrow 1) \cdot o^{\uparrow} T(t_\downarrow 2, t_\downarrow 1) +$$
$$+ \iint_{t1}^{t2} o(t_2 \sigma) \cdot B(\sigma) \psi_W(\sigma) \cdot B^T(\lambda) \cdot o^T(t_2, \lambda) \cdot \delta(\sigma - \lambda) d\lambda d\sigma$$

unde $\delta(t)$ este componenta sistematică a erorii de măsurare, iar ψ_w este o funcție asociată caracteristicii de transfer a sistemului a carui matrice are toate zerourile şi polii situați în semiplanul stâng, ψ_w^T reprezintă conjugata funcției ψ_w.

Dacă efectuăm integrala în raport cu parametrul λ obținem:

$$\left[\sigma_\downarrow \varepsilon^{\uparrow} 2\right)\hat{}(t_\downarrow 2 \mid t_\downarrow 1) = o(t_\downarrow 2, t_\downarrow 1) \cdot \left(\sigma_\downarrow x^{\uparrow} 2\right)\hat{}(t_\downarrow 1) \cdot o^{\uparrow} T(t_\downarrow 2, t_\downarrow 1) +$$
$$+ \int_{t1}^{t2} o(t2\sigma) \cdot B(\sigma) \cdot \psi_W(\sigma) \cdot B^T(\sigma) \cdot o^T(\sigma) \cdot o^T(t_2, \sigma) d\sigma \tag{4.19}$$

Deoarece expresia de sub integrală este pozitiv definită, rezultă că dispersia $[\sigma_1 \varepsilon^{\intercal} 2)\hat{}(t_1 2 \mid t_1 1)$ va creşte nelimitat o dată cu creşterea diferenței dintre momentele de timp t_1 şi t_2. Mai mult, dacă sistemul analizat nu este stabil atunci şi primul termen din membrul drept al ecuației (4.19) va creşte proporțional cu diferența $\Delta t = t_2 - t_1$.

4.1.3 Algoritm de predicție şi estimare a erorilor

Pe baza analizei efectuate şi a relațiilor deduse în paragrafele 4.1.1 şi 4.1.2 se poate imagina algoritmul de predicție şi estimare a erorilor produse la predicția semnalelor de intrare pentru un sistem digital de control adaptiv. Fig. 4.1 prezintă ordinograma algoritmului.

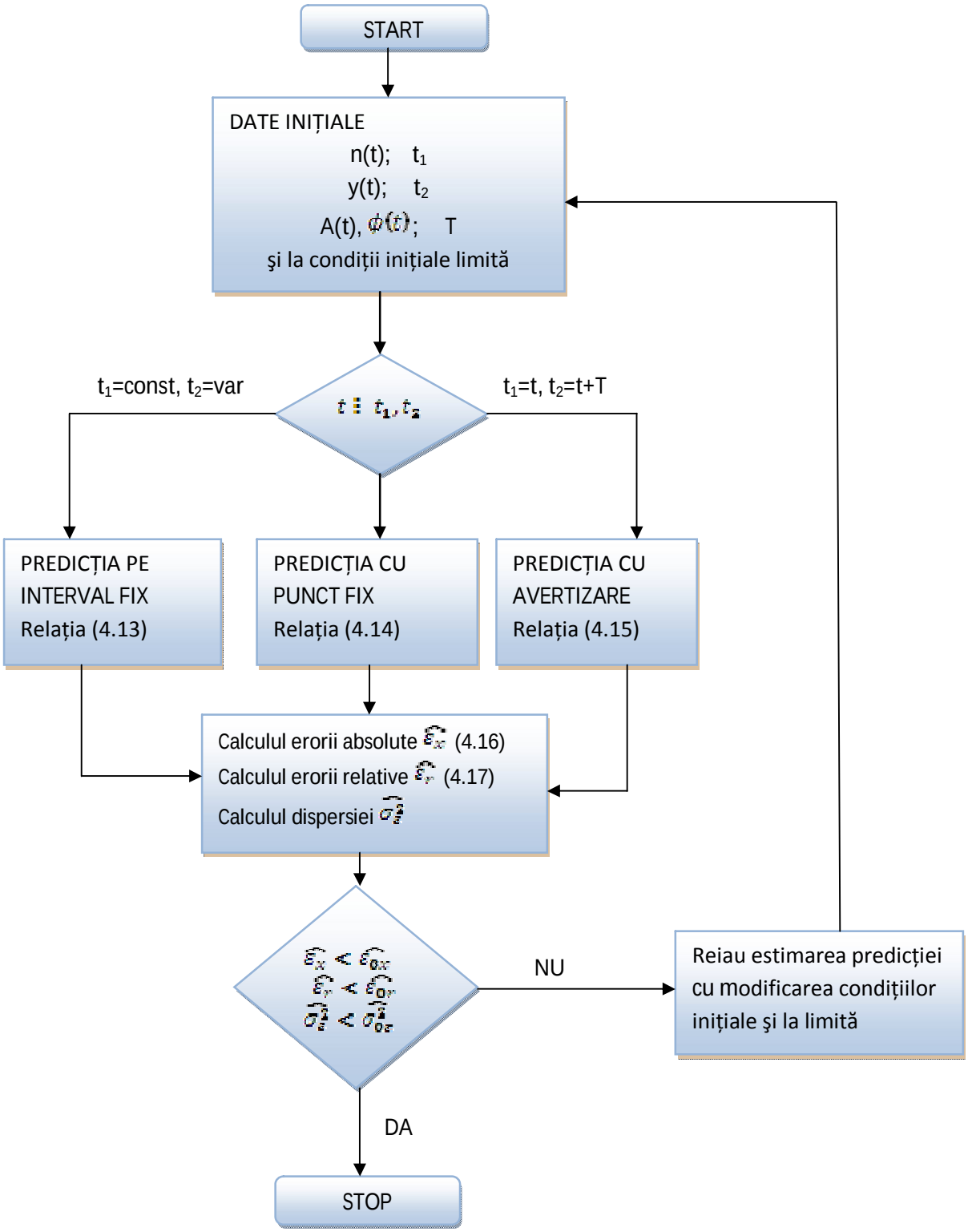


Fig. 4.1 Ordinograma algoritmului de predicție şi estimare a erorilor

4.1.4 Exemplu de aplicație

Vom aplica teoria dezvoltată în cadrul acestei lucrări la studiul sistemului digital de control adaptiv prezentat în Fig.4.2.

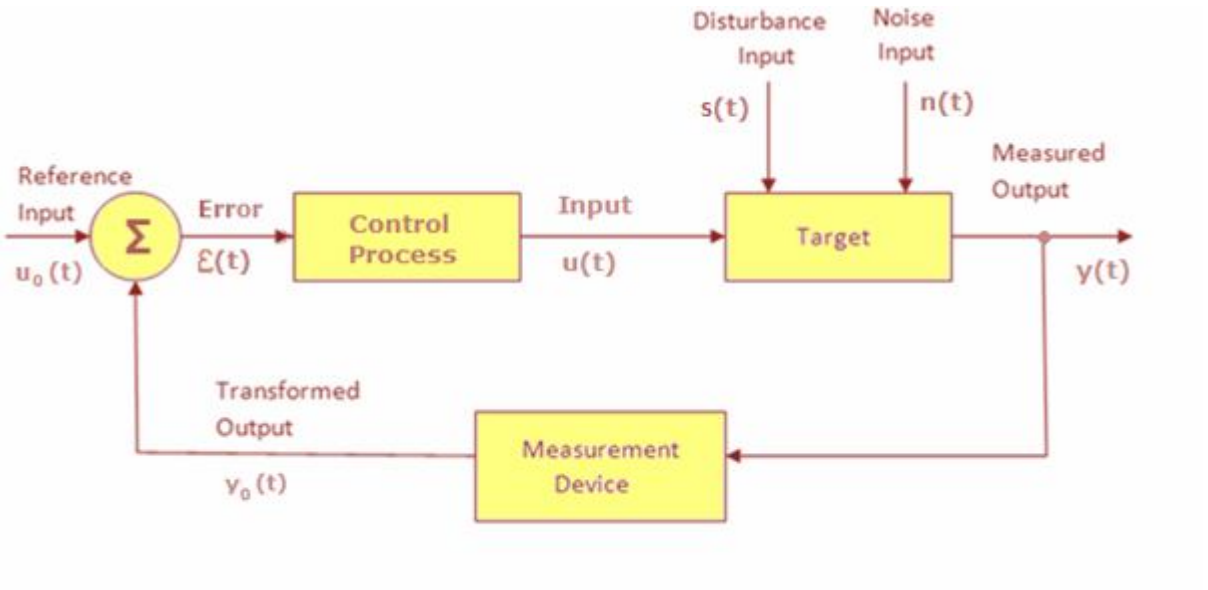


Fig 4.2: Schema de principiu a unui sistem de control adaptiv

Componentele arhitecturii de referință au următoarea semnificație:

- O intrare de referință $u_0(t_0)$
- Eroarea de control : $\varepsilon(t) = \mu_0(t) - y_0(t)$ care reprezintă diferența dintre intrarea de referință şi ieşirea transformată
- Ieşirea măsurabilă y(t)
- n(t) şi s(t) semnale aleatoare care caracterizează zgomotul alb (procese back-up sau virus scan ce acționează pe durata măsurării)
- sistemul omputerizat ce urmează a fi controlat având funcția de transfer w(t)

Presupunem că sistemul de control liniar prezentat poate fi descris de sistemul de ecuații:

$$(A.1)\begin{cases}\frac{du}{dt} = \begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix} n(t) + \begin{bmatrix}0\\ 1\end{bmatrix} w(t)\\ y(t) = n(t) + v(t)\end{cases}$$

Matricea de transfer care caracterizează sistemul dinamic analizat este:

$$\phi(t_2, t_1) = \begin{bmatrix}1 & t_2 - t_1\\ 0 & 1\end{bmatrix}$$

Vom prezenta algoritmul de calcul şi vom calcula dispersia erorii, pentru cele trei forme de predicție descrise în paragraful 4.1.1. Algoritmul general de calcul se poate exprima cu ajutorul relației matriciale:

$$\hat{u}(t_2|t_1) = \begin{bmatrix} 1 & t_2 - t_1 \\ 0 & 1 \end{bmatrix} \hat{u}(t_1), \qquad t_2 \geq t$$

ceea ce este echivalent cu rezolvarea sistemului de ecuații:

$$(A.2)\begin{cases} \hat{u}(t_2 \mid t_1) = (u_1)\hat{}(t_1) + (t_2 - t_1)(u_2)\hat{}(t_1) \\ u(t_2 \mid t_1) = (u_2)\hat{}(t_1) \end{cases}$$

Prin particularizare pentru cele trei tipuri de predicție obținem:

- Algoritmul predicției cu interval fix.

$$(A.3)\left\{ u\hat{}(t \mid t_1) = (u_1)\hat{}(t_1) + (t - t_1)(u_2)\hat{}(t_1) \right. \left. u(t \mid t_1) = (u_2)\hat{}(t_1) \right) \text{ cu } t > t_1$$

- Algoritmul predicției cu punct fix.

$$(A.4)\left\{ u\hat{}(t_2 \mid t) = (u_1)\hat{}(t) + (t_2 - t_1)(u_2)\hat{}(t) \right. \left. u(t_2 \mid t) = (u_2)\hat{}(t) \right)$$

- Algoritmul predicției cu avertizare:

$$(A.5)\left\{ u\hat{}(t_2 \mid t_1) = (u_1)\hat{}(t_1) + (t_2 - t_1)(u_2)\hat{}(t_1) \right. \left. u(t_2 \mid t_1) = (u_2)\hat{}(t_1) \right)$$

Presupunând $\psi_w = const$., algoritmul pentru calculul dispersiei pentru cele trei forme analizate de predicție poate fi scris:

$$(A.6)\left\{ \sigma_{11}^2 \right. \left(t_2 \mid t_1 = \sigma_{11}^2(t_1) + 2(t_2 - t_1)\sigma_{12}^2(t_1) + (t_2 - t_1)^2\sigma_{22}^2(t_1) + \frac{(t_2 - t_1)^3 \psi_w}{3} \right. \left. \sigma_{12}^2(t_2 \mid t_1 \right) = \sigma_{12}^2(t_1) + (t_2 - t_1)\sigma_{22}^2(t_1) + \frac{(t_2 - t_1)^2 \psi_w}{2}$$

Estimarea optimală a predicției se calculează cu relația:

$$\hat{u}(t_1 \mid t) = \phi(t_1, t)\,(u_1)\hat{}(t, t)$$

sau:

$$\hat{u}(t_1) = \phi(t_1, t)\hat{u_1}(t)$$

Se constată aşadar că pentru sistemul de control adaptiv analizat este îndeplinită condiția de estimare optimală (utilizând filtrarea Kalman-Bucy[5]). Ca urmare, este preferabil să se aplice algoritmul de estimare a predicției pe interval fix.

4.2 ALGORITM DE PREDICȚIE PENTRU ESTIMAREA ERORILOR TEMPORARE

Setul de date pe care l-am folosit provine de la un server web de producție aflat la o mare companie de calculatoare. Variabila aleasă spre modelare este numărul de operații de tip HTTP per secundă care sunt realizate de către acest server web. Datele sunt colectate la intervale de cinci minute, adică un total de 288 intervale pe zi.

Datele sunt separate aleator folosind un model "tipare de trafic zilnic" care iau în considerare momentul zilei, ziua din săptamană şi un model de tip autoregresiv folosit pentru a caracteriza procesul rămas. Aici utilizăm acelaşi set de date şi variabile. De asemenea folosim modele separate pentru componentele nestaționare (tendința) şi staționare a procesului original.

Evident, numărul de operatii HTTP per secundă este nestaționar în sensul că media se schimbă odată cu momentul zilei şi ziua din sapatamană. Deci, putem analiza întregul proces ca având o medie stabilă pentru fiecare oră a zilei împreună cu fluctuațiile care sunt modelate de o variabilă aleatoare cu media zero. Tratăm mediile ca un sub-proces separat, la care o să ne referim cu numele de proces tendențial. Scăzând tendința din datele originale, obținem procesul rezidual, care se presupune staționar.

Schema noastră de predicție este structurată după cum urmează: prima dată, procesul tendențial este estimat din cel original. Mai departe, procesul rezidual este construit scăzând tendința din procesul original. Apoi predicția este aplicată procesului rezidual. În final, tendința este adăugată înapoi pentru ultimul rezultat al predicției. Modelarea necesară pentru a susține aceşti paşi este împărțită în două părți: modelarea subprocesului nestaționar (tendința) şi modelarea subprocesului rezidual (rămas).

4.2.1 Algoritm de predicție pentru estimarea ratei de sosire

O săptămână de înregistrări de date a fost utilizată pentru a antrena sistemul de predicție şi să estimeze componentele nestaționare şi staționare ale variabilelor de măsurare.

Filtrarea Datelor

Pentru a elimina zgomotele se foloseşte un filtru Butterworth de ordinul 4. Funcția de transfer pentru un filtru Butterworth trece-jos de ordinul 3 cu frecvența de tăiere de 0.05π rad este:

$$A(q)y(t) = B(q)u(t) + e(t) \quad (4.20)$$

unde :

$A(q) = 1 - 2.686\ q^{-1} + 2.42\ q^{-2} - 0.7302\ q^{-3}$

$B(q) = 0.0004165 + 0.00125\ q^{-1} + 0.00125\ q^{-2} + 0.0004165\ q^{-3}$

Graficele pentru traficul înainte şi după filtrare sunt prezentate în Fig. 4.3:

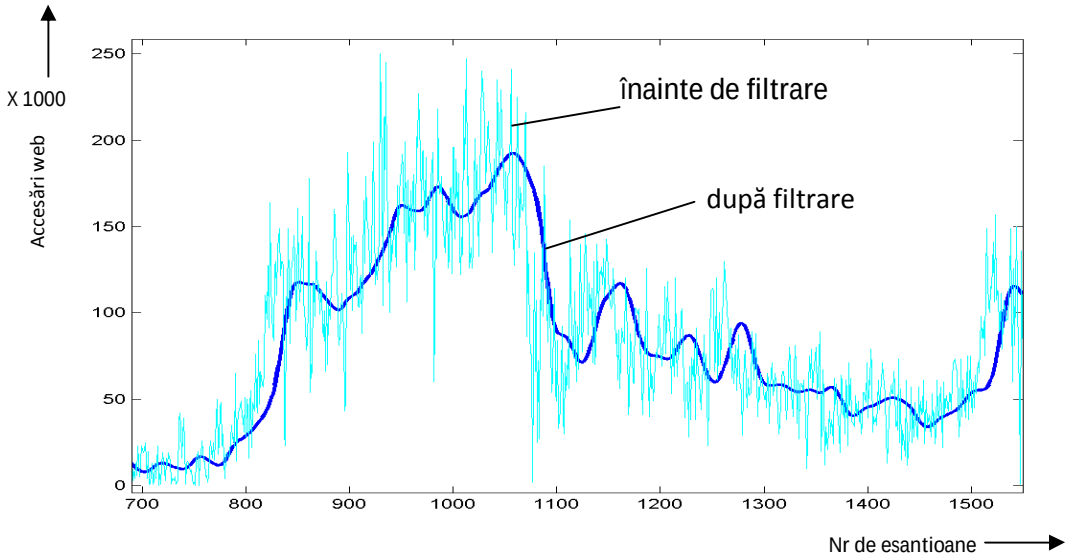


Fig. 4.3 Formele de undă al traficului înainte şi după filtrare

4.2.2 Modelul Estimativ

Modelul este constituit din două submodele: Primul prezintă dependența în timp sau comportamentul nestaționar al variabilei de măsurare. Al doilea submodel este staționar şi analizează dependența temporală a variabilei după ce primul model de analiză a fost izolat.

Un exemplu de comportament nestaționar este prezentat în Fig. 4.4, care trasează graficul dependenței variabilei de măsurare, pe durata a cinci zile de cereri, preluate din serverul web pe care îl studiem.

Se poate observa tendința pronunțată şi consistentă bazată pe momentul zilei.

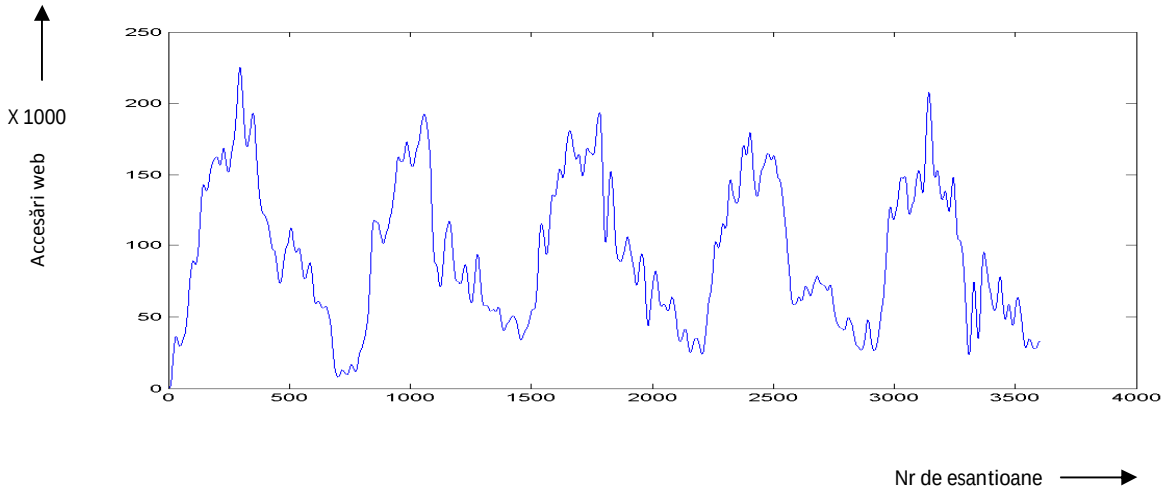


Fig. 4.4 Comportamentul nestaționar al variabilei măsurate

Caracterul periodic al formei de undă se poate determina analizând funcția de autocorelație pentru variabila măsurată (Fig. 4.5).

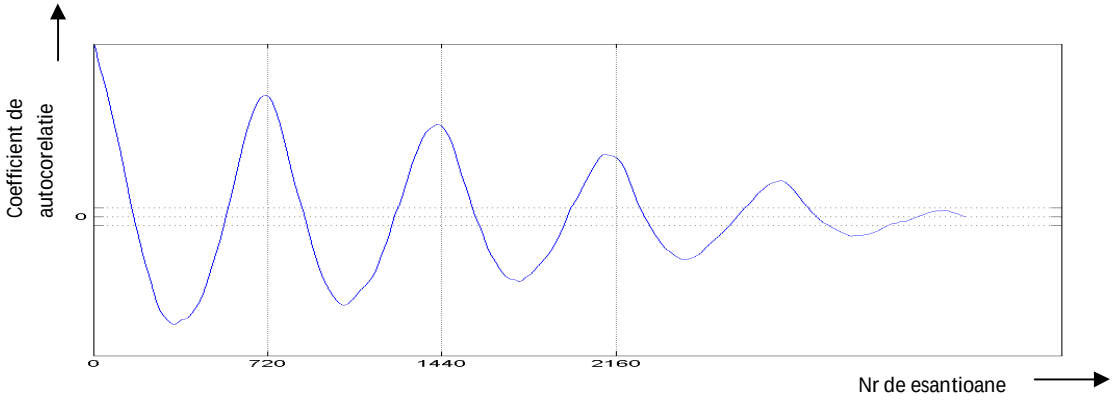


Fig.4.5 Funcția de autocorelație a variabilei

Luăm în considerare momentul zilei, ziua din săptămână şi luna din an. Fie Y_{ij}, variabila aleatoare analizată pentru eşantioanele prelevate la interval de 2 minute din ziua i şi săptămâna j.

- **Metoda utilizată:**

Modelul complet analizează doi factori fără reproducere pentru componenta nestaționară

Se utilizează un sistem autoregresiv liniar de ordinul 10 pentru estimarea tendinței staționare şi dependenței timp-serială.

Modelul pentru abordarea cu doi factori fără reproducere este descris de ecuația:

$$y_{ij} = \mu + \alpha_i + \beta_j + \varepsilon_{ij} \tag{4.21}$$

În această relație, y_{ij} reprezintă valoarea semnalului de ieşire măsurată în experiment între cele două niveluri *i* şi *j*, μ este valoarea medie măsurată, α_i este efectul moment al zilei la nivelul *i*, β_j este efectul zilei din săptămână la nivelul *j*, şi ε_{ij} este eroarea de măsurare. Parametrii α_i şi β_j sunt astfel calculați încât sumele lor sunt zero, adică:

$$\sum_{i=1}^{720} \alpha_i = 0$$

$$\sum_{j=1}^{5} \beta_j = 0$$

Rezultatul măsurătorilor sunt organizate într-o matrice bidimensională de 5 linii (număr de zile) şi 720 coloane (pentru intervalul de eşantionare de două minute rezultă 720 eşantioane pe zi) astfel încât elementul (i,j) corespunde valorii măsurate în experiment (factorul moment al zilei este la nivelul i şi factorul zi a săptămânii este la nivelul j). Coloanele corespund nivelurilor temporale ale zilei, iar liniile corespund nivelurilor ce reprezintă zilele din săptămână.

Valorile parametrilor de model sunt calculate şi alese astfel încât eroarea are o medie egală cu 0. Aceasta înseamnă că suma termenilor eroare împreună cu fiecare coloană şi fiecare linie este 0. În Fig. 4.6 este trasat graficul matricei pentru cinci zile:

Valorile matricei pentru cinci zile:

```
for i=1:5,
   weekMatrix(i,:) = f_week((i-1)*720 + 1:i*720);
end;
```

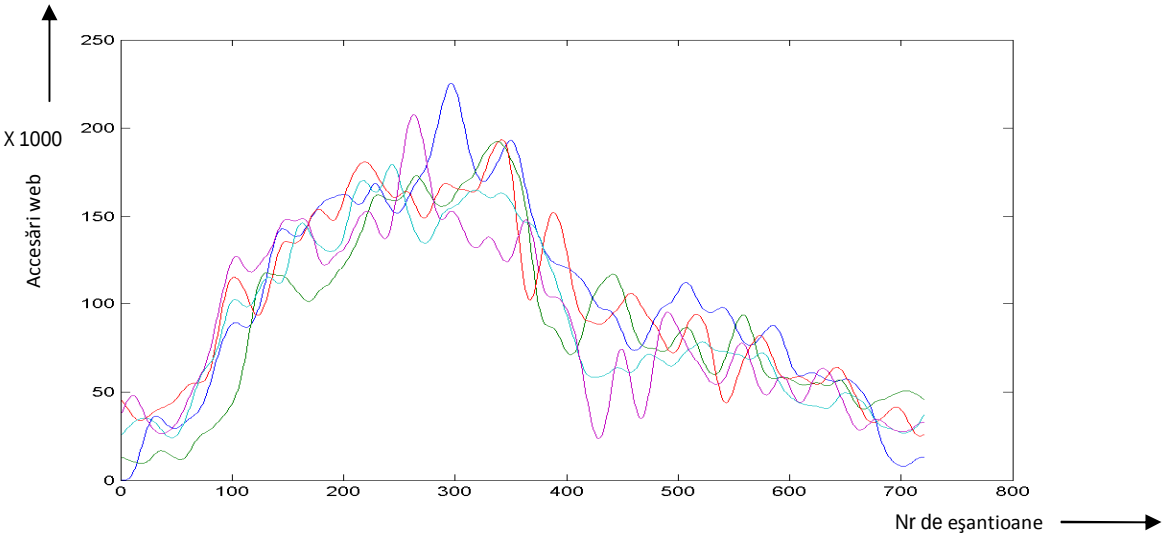


Fig. 4.6 Reprezentarea grafică a variației matricei bidimensionale de date

- **Calculul efectelor:**

Estimările parametrilor de model sunt:

$$\mu = \frac{1}{3600} \sum_{i=1}^{720} \sum_{j=1}^{5} y_{ij}$$

$$\alpha_i = \frac{1}{5} \sum_{j=1}^{5} y_{ij} - \mu$$

$$\beta_j = \frac{1}{720} \sum_{i=1}^{720} y_{ij} - \mu$$

Mediile pe coloane (α_i) caculate din modelul ANOVA sunt prezentate în Fig. 4.7:

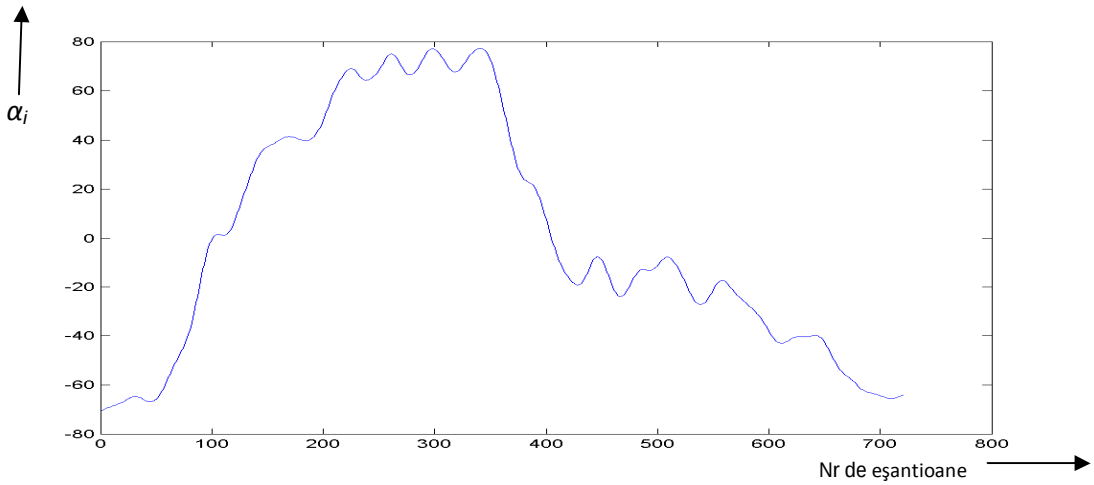


Fig. 4.7 Dependența factorului α_i de numărul eşantioanelor prelevate

```
miuW   = mean2(weekMatrix);
alphaW  = wstats.colmeans - miuW;
betaW   = wstats.rowmeans - miuW;
```

Următoarea formulă caculează eroarea între valorile estimate pentru rata de sosire şi valorile reale, înainte de aplicarea modelului autoregresiv:

```
errorW = f_week - [miuW + alphaW + betaW(1) miuW + alphaW + betaW(2) miuW + alphaW +
        + betaW(3) miuW + alphaW + betaW(4) miuW + alphaW + betaW(5)];
```

Fig. 4.8 prezintă dependența parametruilor f, ε_{ij} de numărul eşantioanelor prelevate precum şi funcțiile de autocorelație corespunzătoare.

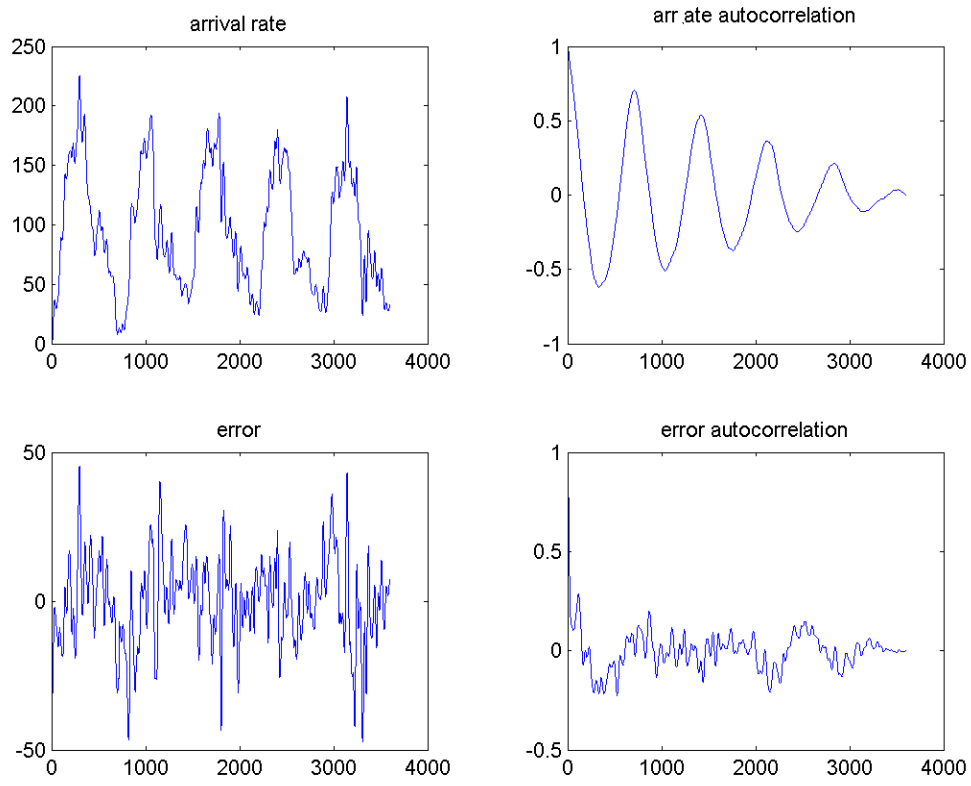


Fig. 4.8 Dependența temporală a parametrilor f şi σ_{ij}

Modelul autoregresiv pentru evaluarea componentei staționare şi dependența timp-serială poate fi scrisă:

ResidModel = ar(m_residW,10);

Ecuația care caracterizează modelul estimat este:

$$A(q)y(t) = e(t)$$

unde:

A(q) = 1 - 1.382 q^-1 + 0.08983 q^-2 + 0.08689 q^-3 + 0.08013 q^-4
+ 0.07033 q^-5 + 0.05866 q^-6 + 0.04633 q^-7 + 0.03436 q^-8
+ 0.02348 q^-9 - 0.1025 q^-10

În Fig. 4.9 sunt reprezentate grafic valorile reziduale şi cele măsurate pentru semnalele la ieşirea unui sistem digital de control predictiv:

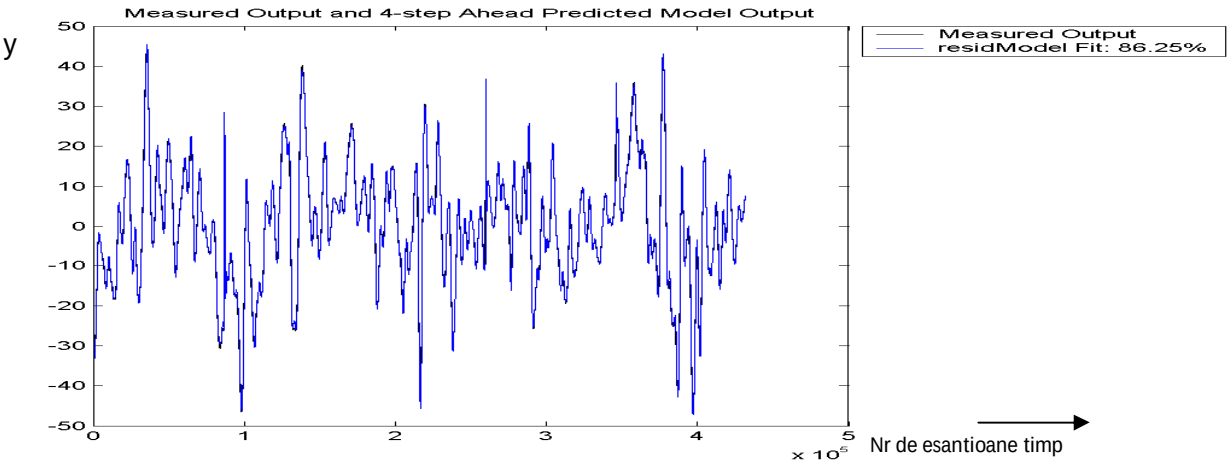


Fig. 4.9 Dependența temporala a valorilor reziduale şi măsurate pentru semnalele la ieşire

Fig. 4.10 prezintă o comparație între semnalele traficului măsurat şi rata de sosire a modelului predictiv:

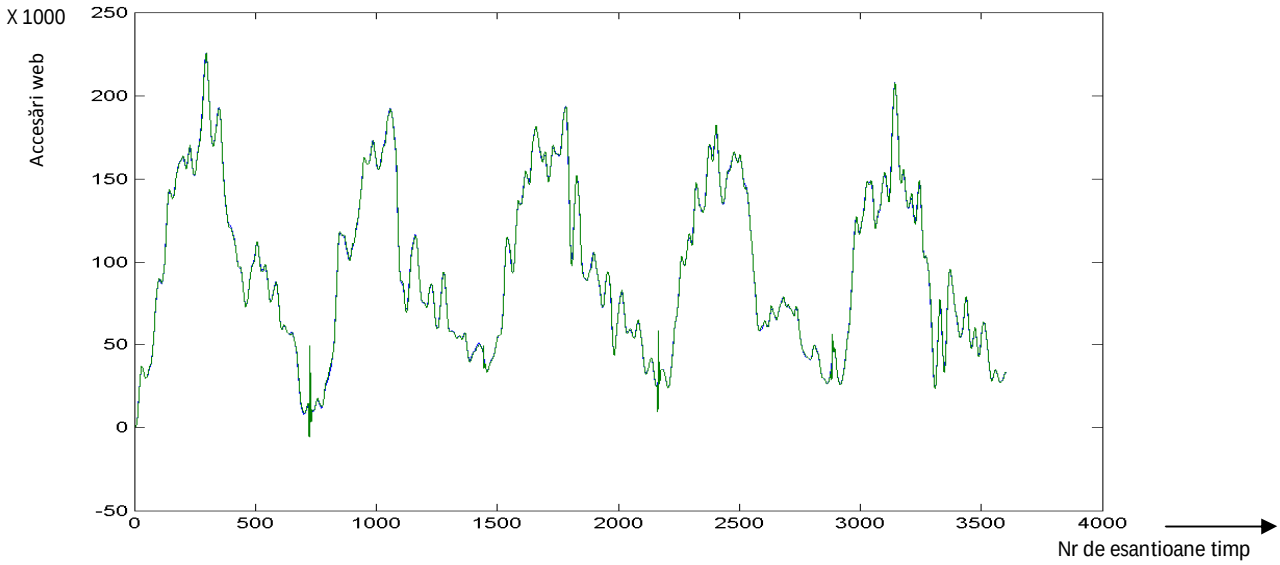


Fig. 4.10 Dependența temporală a traficului măsurat şi a ratei de sosire a modelului predictiv

Din analiza graficului se constată diferențe foarte mici între eşantioanele originale şi valorile estimate de algoritm (practic o singură curbă pe graficul din figura 4.10). În Fig. 4.11 este reprezentată grafică dependența temporal a erorii de predicție.

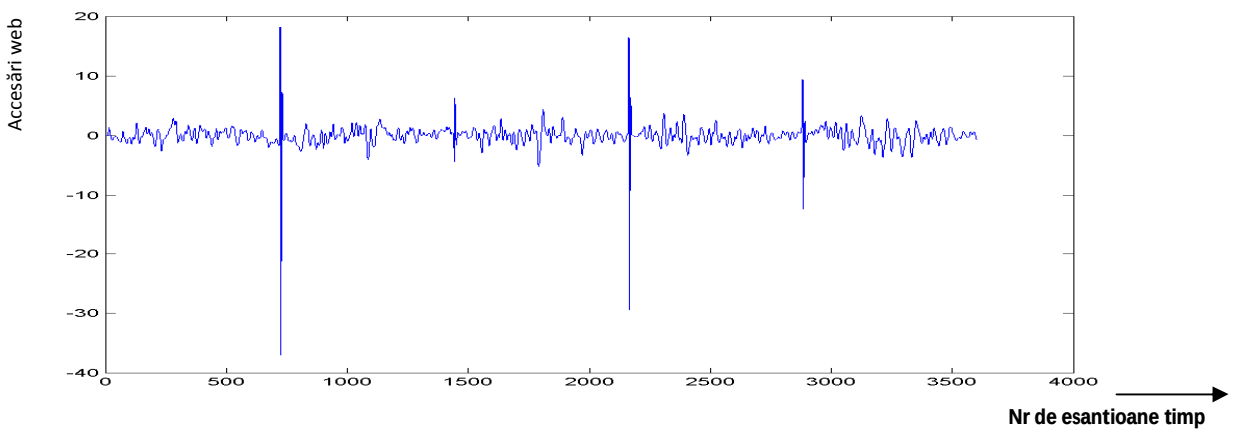


Fig. 4.11 Dependența temporală a erorii de predicție

4.2.3 Estimarea erorii de predicție

Am utilizat două tehnici pentru modelarea comportamentului nestaționar al mediei unei matrice (rata de sosire). Analiza procesului staționar (dependența timp-serial) se realizează după ce efectele comportamentului nestaționar au fost izolate. Bazându-ne pe aceste două modele şi presupunând că semnalele u(t) sunt independente şi identic distribuite, încercăm să estimăm probabilitatea trecerii peste un prag (de exemplu,încălcarea obiectivului de performanță).

Ecuația pentru estimarea ratei de sosire poate fi scrisă:

$$ar = \mu + \alpha_i + \beta_j + \gamma_k + residW \qquad (4.22)$$

A fost utilizat următorul model autoregresiv (AR2) pentru a elimina dependențele timp-seriale (care reprezintă componenta staționară *residW*) :

$$\begin{cases} residw = \text{EMBED Equation.3}\, y(t-1) + \text{EMBED Equation.3}\, y(t-2) + u(t) \\ y(t) = \text{EMBED Equation.3}\, y(t-1) + \text{EMBED Equation.3}\, y(t-2) \end{cases} \qquad (4.23)$$

În relația (4.23), u(t) reprezintă variabile aleatoare independente şi identic distribuite cu media egală cu zero, σ_u^2 este dispersia parametrului u(t) şi ϕ_1, ϕ_2 sunt parametrii modelului estimat în timpul procesului de antrenare. Valorile pentru cei doi parametri sunt:

ϕ_1 = 1.494

ϕ_2 = -0.4976

Utilizând relația (4.19) se poate calcula estimatorul nebalansat al erorii de dispersie σ_u^2 :

$$\sigma_u^2 = 0.6590$$

Pentru a verifica dacă este rezonabil să presupunem că elementele vectorului u(t) sunt normal distribuite, noi trasăm quantilele standardului normal cu quantilele distribuite empiric pentru u(t) în date de tip HTTP. Caracterul liniar pronunțat al graficului din Fig. 4.12 sugerează că presupunerea unei distribuții normale este rezonabilă.

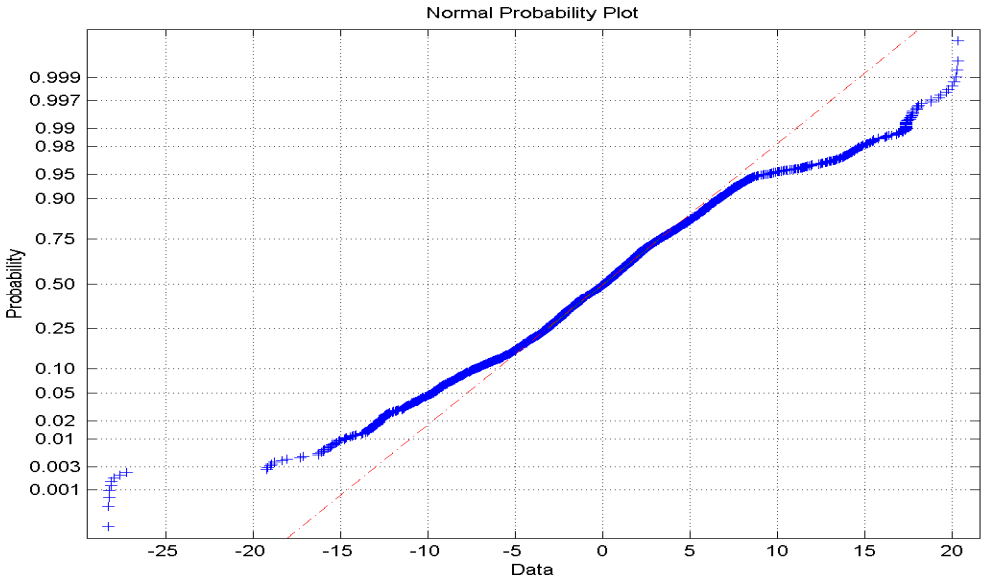


Fig. 4.12Dependența probabilității de estimare a erorilor în funcție de datele eşantionate

Distribuția normală ideală cu media egală cu zero şi dispersia σ_u^2 împreună cu distribuita empirică a erorii u(t) sunt reprezentate în grafic în Fig. 4.13.

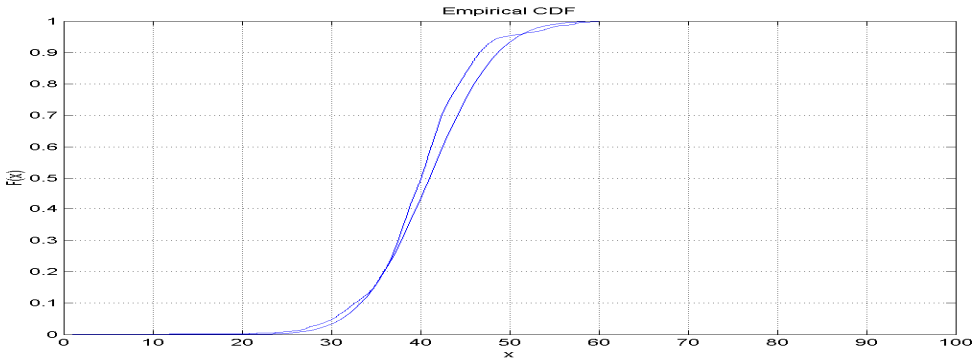


Fig. 4.13 Distribuția normal, ideală şi empirică a erorii

Timpul curent este t. Valorile la t-1 şi t-2 reflectă trecutul recent şi dorim să estimăm probabilitatea trecerii peste un prag TR la timpii t+1, t+2, t+3, situați în viitor.

Modelul comportamentului nestaționar a fost utilizat să transforme componentele variabile în timp ale lui f_week (f_săptămână) în residW staționar. Fiindcă transformăm măsurarea, trebuie să transformăm şi pragul TR în aceeaşi manieră. Un prim efect îl constituie dependența temporal a pragurilor transformate. Dacă tr(t) este valoarea transformată a lui TR la momentul de timp t.

$$tr(t) = TR - \mu - \alpha_i - \beta_j - \gamma_k = TR - nonstatW \quad (4.24)$$

Din ecuația (4.23) putem estima y(t+h), care denotă o predicție care se află cu h momente de timp în viitor, unde h este orizontul de predicție. Din cauza variațiilor aleatoare, y(t+h) pot fi mai mari decât tr(t+h), dar valoarea reală a lui y(t+h) poate fi mai mică decat tr(t+h). Din acest motiv trebuie să calculăm Pt(h) ca probabilitate a evenimentului y(t+h) pentru a depăşi pragul ținând cont de valorile cunoscute ale lui y(t) şi y(t-1). Pt(h) înglobează informații atât despre valoarea aşteptată şi despre dispersia lui y(t+h) condiționată de observațiile în timp t.

Odată ce am estimat valorile aşteptate ale măsuratorilor viitoare prin cacului lui y(t+h), mai trebuie să calculăm dispersia estimată a unei măsuratori viitoare fiind date y(t), y(t-1). Începem prin exprimarea lui y(t) ca o funcție de u(t). Folosind operatorul de întârziere q ($q^m X(t)=X(t-m)$) scriem ecuația (4.23) sub forma:

$$(1 - \phi_1 q - \phi_2 q^2)y(t) = u(t) \quad (4.25)$$

adică,

$$y(t)= \frac{u(t)}{(1-a_1 q)(1-a_2 q)} \quad (4.26)$$

unde a_1, a_2 sunt solutiile ecuației $(1 - \phi_1 q - \phi_2 q^2) = 0$, adică:

$a_1 = 0.5014$

$a_2 = 0.9923$

Dispersia $\sigma_u^2(t+h)$ poate fi calculată acum cu relația:

$$\sigma_u^2(t+h) = \sigma_u^2\left[\sum_{n=0}^{h-1} \frac{(a_1^{n+1} - a_2^{n+1})^2}{(a_1 - a_2)^2}\right] \quad (4.27)$$

unde σ_u^2 este acelaşi cu cel din ecuația ($\sigma_u^2 = 0.6590$).

În Fig. 4.14 este reprezentată grafic dependența temporală a dispersiei estimată pentru parametrul u(t):

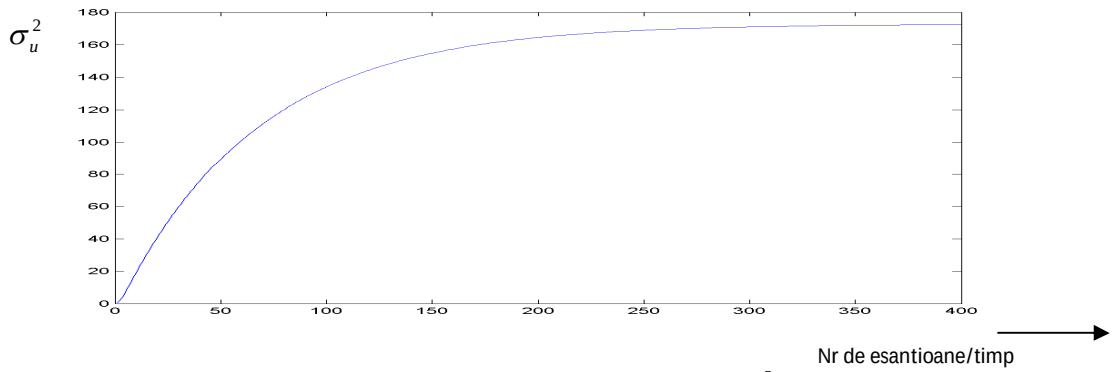


Fig. 4.14 Dependența temporal a dispersiei σ_u^2

Deoarece dispersia semnalului residW este var(y(t))=172.13 se poate constata analizând Fig. 4.14 că dispersia lui u(t), $\sigma_u^2(t+h)$ converge către dispersia lui y(t), care este o valoare staționară pentru $h \to \infty$.

Pe baza studiului inițiat, se constată că probabilitatea ca semnalul estimat să fie situat între două praguri TR_s (pragul superior) şi TR_i (pragul inferior) este:

$$P(t+h) = normcdf\left(\frac{y(t+h) - tr_s(t+h)}{\sigma_u(t+h)}\right) + normcdf\left(\frac{tr_i(t+h) - y(t+h)}{\sigma_u(t+h)}\right) \tag{4.28}$$

În această relație normcdf(x) reprezintă funcția cumulativă de distribuție pentru standardul normal, $tr_s(t+h)$, $tr_i(t+h)$ reprezintă pragul superior respectiv inferior transformat. Distribuția de probabilitate pentru diferite valori ale orizontului de predicție h sunt reprezentate grafic în Fig. 4.15:

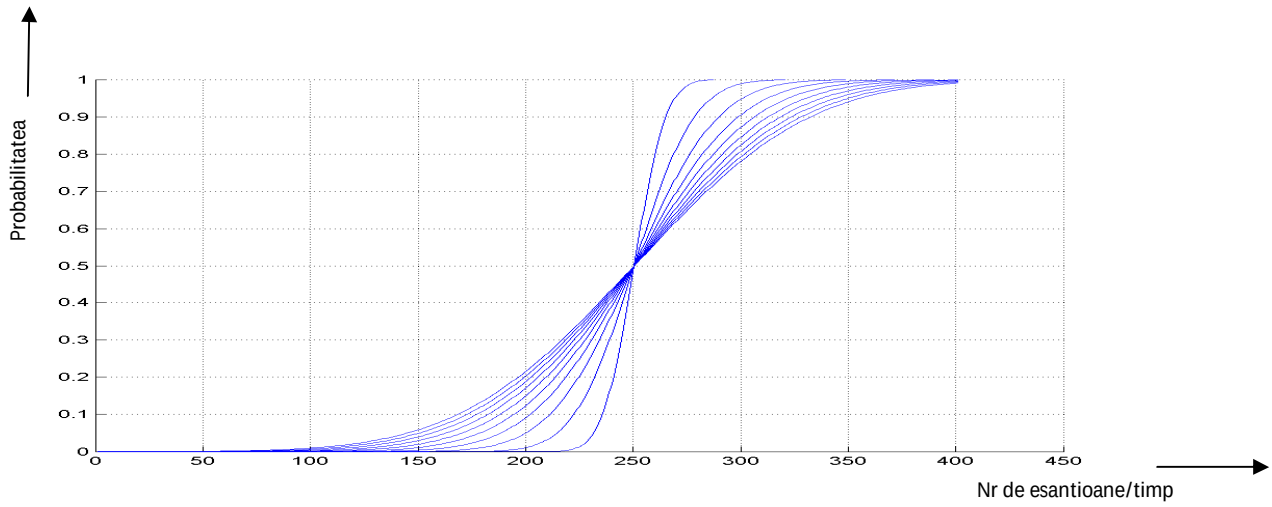


Fig. 4.15 Dependența temporală a probabilității orizontului de predicție

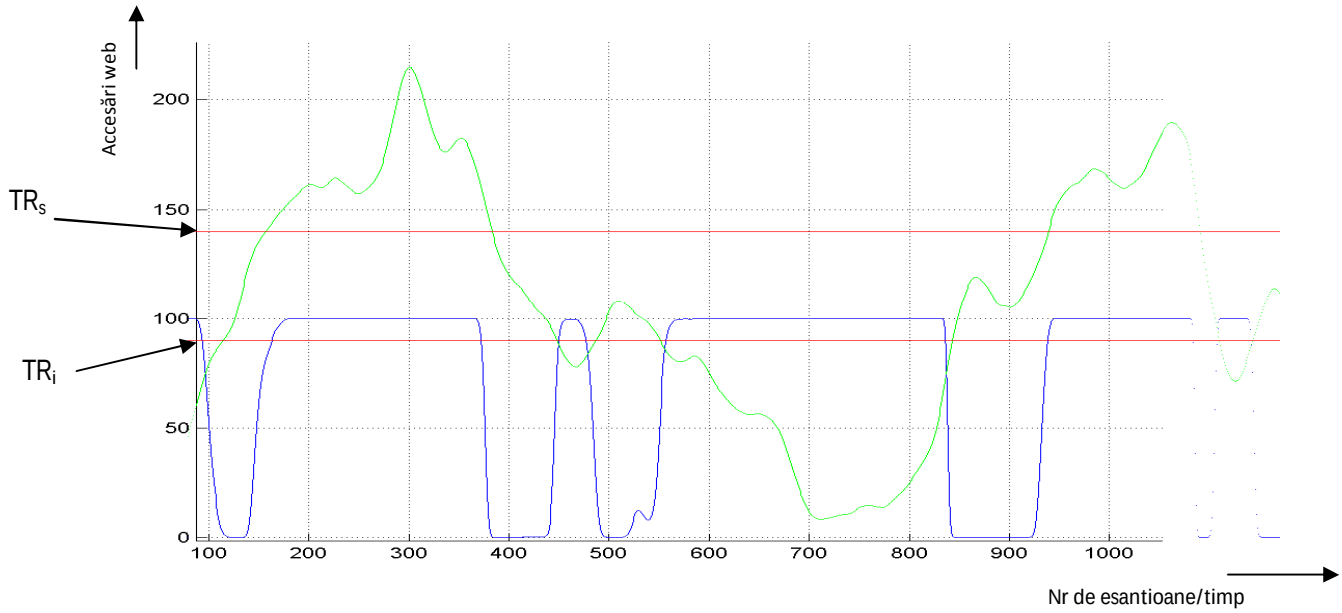


Fig. 4.16 Dependența temporală a densității de probabilitate şi a traficului măsurat

În Fig. 4.16 am reprezentat pe acelaşi grafic dependența temporal a funcției densității de probabilitate, considerând un set de praguri TR_s=140 si TR_i=90. Linia albastră reprezintă densitatea de probabilitate, iar cea verde, traficul măsurat.

Pentru a putea reprezenta probabilitatea P(t+h) ca funcție de numărul de servere din cluster(grup de servere) la un moment t+h, este nevoie de o nouă transformare, bazându-ne pe relația dintre cpu şi rata de sosire:

$$N(t+h) = \frac{ar(t+h)}{sla_{cpu}(t+h)} \frac{cpu(t)}{ar(t)} N(t) \tag{4.29}$$

unde: N este numărul total de servere, *ar* este rata de sosire, sla_{cpu} este utilizarea dorită a cpu–lui, iar cpu(t) este utilizarea actuală a cpu –lui. Transformarea pragului din TR_s la un prag resursă (număr de servere din mănunchi) se realizează cu relația:

$$TR_s(t+h) = \frac{N(t+h)}{N(t)} sla_{cpu}(t+h) \frac{ar(t)}{cpu(t)} \tag{4.30}$$

Folosind rezultatele prezentate în relațiile (4.24), (4.28) şi (4.29) se obține expresia funcției probabilitate a distribuției erorilor.

4.3 CONCLUZII ŞI CONTRIBUȚII

Sistemele digitale de calcul autonom s-au dezvoltat în ultimii ani pe structura sistemelor inteligente de control capabile să gestioneze atribute noi precum: autoconfigurarea, autoprotecția, autorepararea sau autooptimizarea. Toate semnalele aplicate la intrările sistemelor autonome de calcul sunt afectate de „zgomote". Ele reprezintă semnale deterministe sau aleatoare, nepurtătoare de informație, care pentru anumite valori ale amplitudinii, sau spectrului de frecvență, pot determina o comportare incorectă a sistemului, prin alterarea informației utile. Estimarea proceselor aleatoare se realizează de regulă prin filtrare, interpolare şi predicție. Este cunoscut că asupra sistemelor reale acționează simultan semnale aleatoare şi deterministe, iar observațiile şi măsurătorile se efectuează pe un interval de timp finit. Ca urmare, este firesc să se utilizeze la sinteza sistemelor autonome de calcul optimal, estimația semnalelor probabile de intrare, utilizând elemente de teoria statistică a deciziilor.

În scopul prelucrării şi controlului unei cantități minime de informație, am inițiat un studiu privind implementarea algoritmilor de predicție în arhitectura sistemelor autonome de calcul şi am evaluat erorile de modelare produse la predicție. Principalele probleme abordate şi soluționate în cadrul acestui capitol sunt următoarele:

- Predicția proceselor aleatoare ca soluție eficientă de analiză şi control a parametrilor de proces ai sistemelor digitale de calcul autonom.
- Analiza efectuată are la bază constatarea că zgomotul util şi zgomotul prezent la intrările sistemului, sunt procese aleatoare staționare necorelate, deci se poate analiza separarea semnalului util de zgomot prin procedee de filtrare adaptivă.
- Estimarea predicției proceselor aleatoare, a generat suportul matematic pentru identificarea şi definirea celor trei tipuri de predictie: pe interval fix, cu punct fix şi cu avertizare.
- O atenție deosebită a fost acordată estimării erorilor produse la predicție, luând în considerare erorile de modelare, intervalul de estimare şi dispersia zgomotului prezent la intrare.
- Pe baza formalismului matematic dezvoltat, s-a elaborat un algoritm flexibil de evaluare a predictiei şi de estimare a erorilor aplicabil, sistemelor autonome de calcul.
- A fost dezvoltat un algoritm de predicție pentru estimarea erorilor temporale care apar la analiza unui set de date provenite de la un server Web, de la o companie mare de calculatoare.
- Datele achiziționate pe durata unei săptămâni de înregistrări continue au fost utilizate în procesul de învățare a sistemului de predicție propus pentru estimarea componentelor nestaționare şi staționare al variabilelor măsurate.

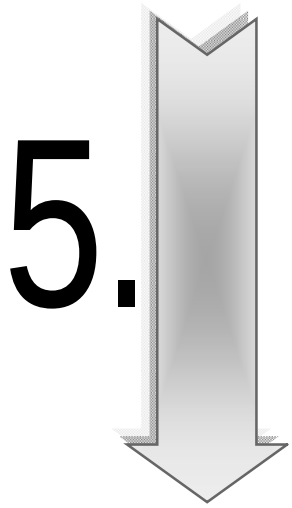
- Graficele trasate sunt intuitive şi ddemonstrează că erorile produse la predicție se încadrează între pragurile limită impuse

Contribuțiile originale aduse la elaborarea acestui capitol pot fi sistematizate astfel:

- O abordare unitară a problemelor ridicate de sinteză a sistemelor autonome de calcul utilizând elemente de teoria statistică a deciziilor
- Implementarea algoritmilor de predictie in arhitectura sistemelor autonome de calcul şi evaluarea erorilor de modelare produse la predicție.
- Prelucrarea şi controlul unei cantități minime de informații prin estimarea şi predicția semnalelor aleatoare prezente la intrările sistemelor digitale autonome de calcul.
- Dezvoltarea suportului matematic pentru estimarea şi predicția zgomotelor, analizate ca şi procese aleatoare staționare.
- În funcție de momentele de timp la care se prelevează probe din semnalele am identificat trei tipuri generice de predicție: pe interval fix, cu punct fix şi cu avertizare.
- Evaluarea erorilor de modelare produse la predicție în ipoteza că intervalul de estimare este bine definit şi dispersia zgomotului prezent la intrare se situează într-o bandă limitată
- S-a demonstrat că dispersia erorii de estimare a procesului de predicție creşte nelimitat odată cu creşterea diferenței dintre momentele de timp la care se fac măsurătorile.
- Pe baza suportului matematic dezvoltat, s-a imaginat un algoritm de predicție şi estimare a erorilor, aplicabil la optimizarea sistemelor digitale de control adaptiv.
- Aplicarea relațiilor deduse pentru cele trei tipuri de predicție, la studiul concret al unui sistem de control adaptiv, a demonstrat că poate fi îndeplinită conditia de estimare optimală cu erori minime, dacă se aplică algoritmul de predicție pe interval fix.
- Elaborarea modelului estimativ pentru datele înregistrate pentru o săptămână a furnizat informațiile necesare procesului de antrenare prin învățare a sistemului de estimare prin predicție.
- Au fost trasate grafice intuitive care ilustrează dependența temporală a parametrilor modelului de estimare propus.

Calculele analitice şi simulările au condus la concluzia că dispersia erorilor de estimare se încadrează în limitele prestabilite.

Pe baza studiului inițiat în acest capitol, se poate concluziona că utilizarea algoritmilor de predicție pentru estimarea erorilor reprezintă o soluție viabilă de creştere a performanțelor şi scădere a costurilor sistemelor autonome de calcul digital.



5. REZULTATE EXPERIMENTALE

Toate dezvoltările teoretice prezentate în capitolele anterioare au putut fi verificate din punct de vedere al eficienței şi implicit al costurilor solicitate, utilizând echipamentele disponibile în cadrul laboratoarelor de cercetare ale IBM Toronto, Canada şi Somers, New York, SUA. În cadrul acestui capitol vom prezenta rezultatele experimentale obținute în cadrul a cinci aplicații dedicate folosind bucle autonome de calcul implementate în IBM – TIO (Tivoli Intelligent Orchestrator Server).

Rezultatele experimentelor au confirmat importanța şi acuratețea dezvoltărilor matematice anterioare, iar simulările au permis rularea buclelor de control autonom suprapuse cu bucla de control adaptabilă.

Scopul urmărit îl constituie optimizarea numărului de servere din cluster care participă în procesele de prelucrare şi de trafic pentru reducerea timpilor de prelucrare urmărind abilitatea sistemelor autoadministrate de a răspunde dinamic la schimbările de obiectiv.

5.1 APLICAȚE WEB AUTOADMINISTRATĂ

- **Arhitectura şi testarea sistemului**

Figura 5.1 prezintă arhitectura propusă pentru sistemul autoadministrat şi paturile de testare utilizate.

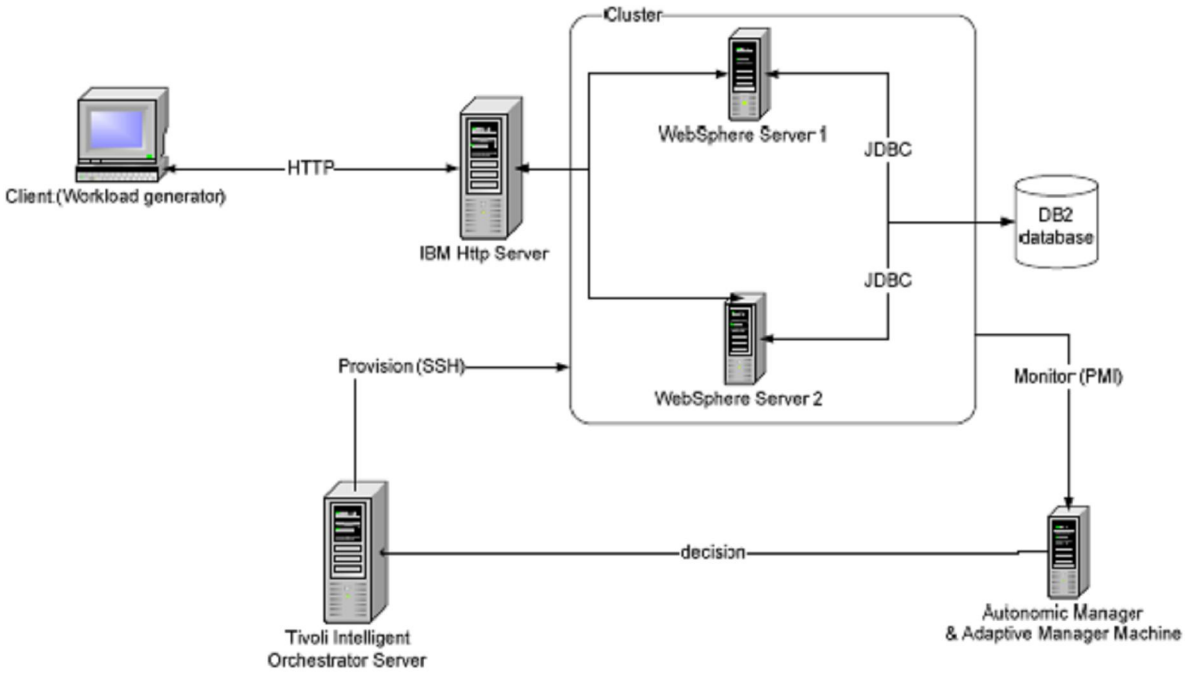


Fig. 5.1 Arhitectura sistemului autoadministrat

Un grup de doua IBM "WebSphere Application Servers" este utilizat pentru gestionarea datelor sistemului administrat. Numărul de calculatoare (dimensiunea grupului) este adjustat dinamic de către bucla autonomă de calcul implementată în IBM Tivoli Intelligent Orchestrator Server (TIO). TIO foloseşte un grup comun de servere pentru a adăuga sau înlătura servere de la grupul de lucru. Acest grup este în mod potențial partajat astfel încât să fie util şi pentru alte aplicații. Prin aceasta se reduce semnificativ numărul de servere necesare pentru un centru de calcul. Un IBM HTTP Server este folosit pentru a distribui traficul de Web al clientului/utilizatorului alocat grupului de servere. Pentru acest pat de testare, strategia folosită pentru distributia de trafic/cereri http este "Round Robin". Cererile sunt alocate diferențial la serverele din grup [BS-08b].

Toate calculatoarele sunt de tip Intel Pentium 4 având f = 3.4 GHz cu 2 GB RAM functionând cu sistemul de operare Microsoft Windows XP Service Pack 2. Fiecare calculator dispune de două CPU şi utilizarea individuală a unui calculator este proiectată să optimizeze media performanțelor celor două CPUs. Aplicația serverelor ce au format grupul incarca IBM Web-Sphere Application Server 6.0 cu Java™ Platform, Enterprise Edition folosind J2SE 1.4.

Grupul de servere din Fig. 5.1 reprezintă sistemul de calcul administrat autonom, iar calculatorul TIO îndeplineşte rolul de control autonom. Bucla adaptabilă de control functionează ca un procesor separat pe acelaşi server ca şi TIO.

Clientul este un generator de trafic virtual care modifică rata de emisie periodic utilizând o distribuție liniara si o distributie Gaussiana. Fiecare client reprezintă o componentă separată ce generează o cerere http URL pentru o adresă dată.

Aplicația este controlată de principiile şi necesitățile comerțului electronic distribuit, exemplu WebSphere Application Server furnizat de către IBM. Aplicația permite ca toate cererile să fie generate pe servlet şi acesta simuleaza diferite actiuni disponibile în aplicație. Distributia liniară simulează o perioada de "stand-by" când rata de sosire a cererilor utilizatorului şi durata de lucru sunt constante. Distribuția Gaussiana simulează o perioada de timp în care rata de sosire a cererilor utilizatorului şi durata sesiunii de lucru au valori cu o distribuție aleatoare.

Senzorul în timp real care gestionează bucla de control este o clasă Java ce face o verificare periodică a celor doua servere, folosind "Performance Monitoring Infrastructure" (PMI) implementată pe serverele WebSphere. Sensorul monitorizeaza patru variabile care sesizează încărcarea serverului:

- Utilizare CPU
- Timp de raspuns servlet
- Incarcarea servlet
- Numarul de clienti ce acceseaza serverul

Când aplicația solicită servere multiple în grup, valoarea primelor trei variabile va fi calculată ca media serverelor, iar numarul de clienti conectați poate fi evaluat ca suma cererilor de accesare a serverelor.

Pentru experiment au fost utilizați doi controleri, iar administratorul adaptabil autonom are menirea să comute între cei doi controleri pe baza distributiei create de generatorul de cereri sintetice. În cazul în care distribuția încărcării lucrului este liniară este folosit un controler ce adaugă sau înlătură servere bazate pe minimum sau maximum de praguri folosite de către CPU.

Când distributia este Gaussiana, controlerul utilizează un filtu Kalman. Controlerul foloseşte informația măsurată pentru a estima timpul de răspuns al aplicației şi stabileşte cel mai mic număr de servere ce va menține răspunsul de ansamblu al sistemului, între valorile de prag prestabilite.

Mecanismul de acționare este bazat pe TIO. Când controlerul ia decizia de înlăturare sau adăugare a unui server din grup, aceasta este transmisă la TIO. Atunci transferă responsabilitatea lui TIO de a decide ce server va fi adăugat sau înlăturat. Odată ce schimbarea serverului este terminată, informația este transmisă înapoi la controler pentru a actualiza modelul sistemului administrat.

- **Rezultatele experimentului**

Bucla de control a fost programată pentru diferite configurații scopul urmărit fiind adăugarea / înlăturarea serverului din sistem.

În primul test efectuat, controlerul care ia decizii bazate pe un prag prestabilit, se utilizează pentru a menține grupul de servere la o valoare optimală. Pragul controlerului funcționează: când utilizarea CPU ajunge la 60%, un server este adăugat şi când utilizarea trece sub 50% un server este înlăturat. Încărcarea lucrului este caracterizată de către distribuția Gaussiană cu 600 de utilizatori şi cu valoare tipică/standard de 400 utilizatori. Deoarece distribuția este Gausssiană, valoarea generată aleator are probabilitatea scăzută să depăşească trei deviații standard.

Când se generează valori mici de utilizatori variabilele analizate sunt pozitive. Fig. 5.2 prezintă dependența densității de numărul utilizatorilor conectați la sistem. Fig. 5.3 reprezintă distribuția temporală a utilizatorilor. Fig. 5.4 reprezintă timpul de răspuns al servletului, Fig. 5.5 utilizarea CPU ca o funcție de timp, iar Fig. 5.6 arată numărul de servere dintr-un grup(cluster) în funcție de timp.

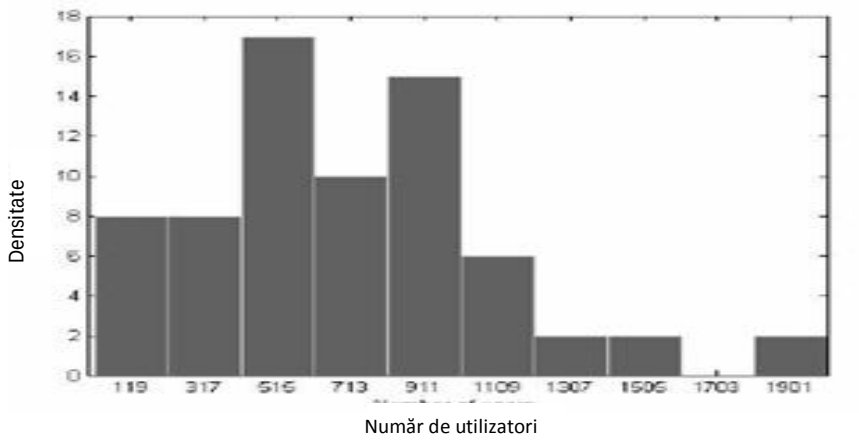


Fig. 5.2 Dependența densității de numărul utilizatorilor

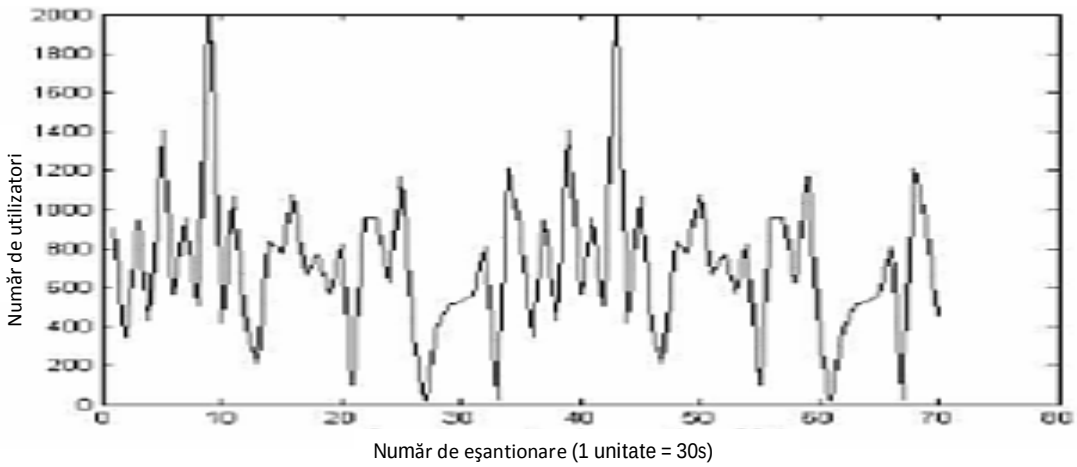


Fig. 5.3 Distribuția temporală a utilizatorilor

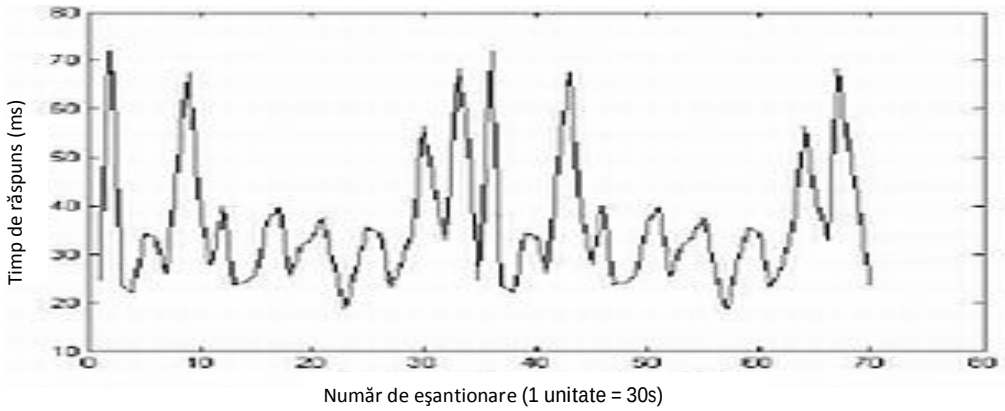


Figura 5.4 Dependența timpului de răspuns

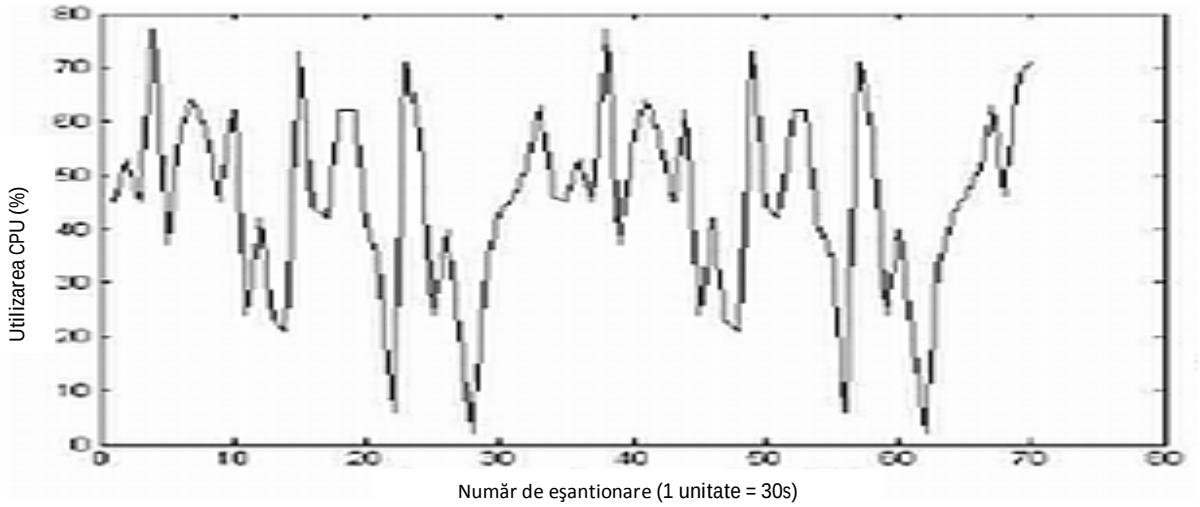


Fig. 5.5 Utilizarea CPU

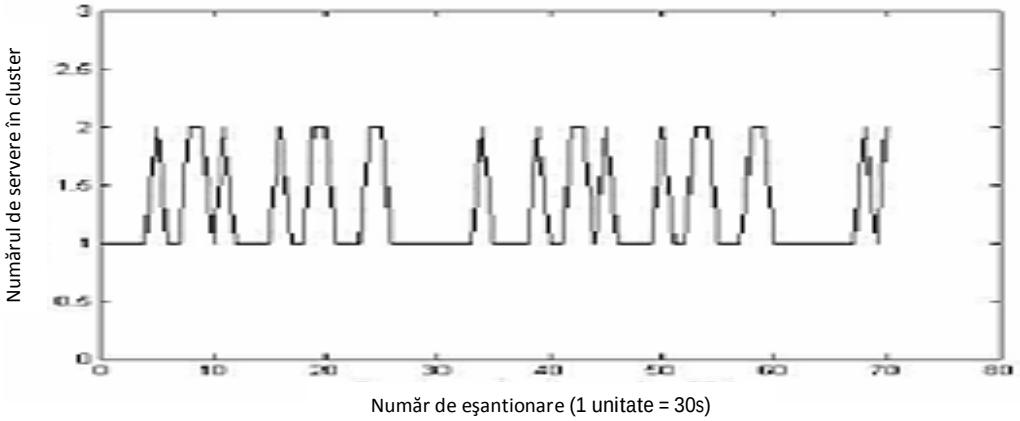


Fig. 5.6 Servere în cluster

5.2 CONTROLUL AUTOMAT AL SERVERELOR INTR-UN SISTEM GRID

În general, un sistem autonom este definit prin existența a patru module care îndeplinesc funcțiile de monitorizare/supraveghere, analiză, planificare şi execuție. Scopul unui sistem autonom este de a automatiza şi simplifica administrarea resurselor de calcul într-un centru de calcul [BS-07c].

Într-un sistem grid se pot defini două clase de lucru, şi anume aplicații interactive, pentru care parametrul caracteristic este timpul de răspuns, definit pentru un anumit procent de utilizatori (de exemplu 99%) şi aplicații tip batch, pentru parametrul caracteristic este timpul necesar pentru execuție, în general definit ca un obiectiv de termen limită de completare.

În experimentul descris, am definit un ansamblu de testare, un model analitic pentru simularea şi estimarea probabilității de încălcare/depăşire a valorilor admisibile pentru obiectivele de serviciu. Modelul de control este astfel conceput încât să personalizeze pentru fiecare clasă de calculatoare condițiile în care se poate adăuga sau extrage un calculator din grupul alocat. Ansamblul de testare a fost folosit pentru estimarea algoritmilor de control.

Elementul de control folosit a fost "Tivoli Intelligent Orchestrator" (TIO), care automatizează procesul de a adăuga sau înlătura un server bazat pe calcularea unui obiectiv de performanță.

TIO include un modul numit Analizator de Obiectiv (Objective Analyzer OA), care îndeplineşte funcțiile de monitorizare a resurselor pentru a identifica decizia folosirii acestora în aplicații, de prescriere a cererilor http, de estimare şi raportare a probabilității de încălcare a obiectivelor de serviciu.

Schema de principiu a unui sistem de tip grid este prezentata in Fig. 5.7.

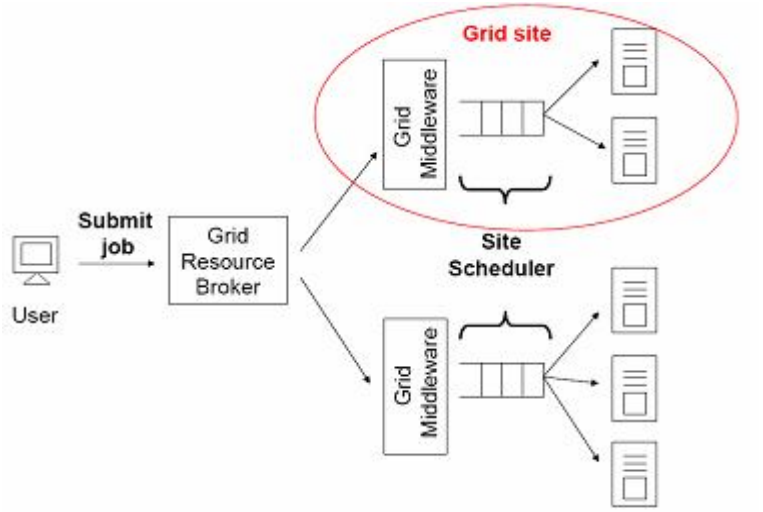


Fig. 5.7 Schema unui sistem de tip grid

Asa cum s-a descris mai sus, un sistem de tip grid este folosit atât pentru încărcări de tip interactiv cât şi pentru încărcări de tip batch. În acest experiment, am folosit încărcări de tip "batch", caracterizate prin un timp de rulare lung, necesită să fie executate înainte de un termen limită şi de obicei folosesc la maximum CPU al calculatorului pe care rulează.

În scenariul care s-a implementat, am presupus că încărcările (programe de lucru) de tip interactiv şi "batch" sunt alocate la grupuri diferite de calculatoare, în condițiile în care cele două grupuri folosesc acelaşi grup de resurse comune pentru adăugare şi înlăturare de resurse (calculatoare). Toate programele sunt pornite înainte de un termen limită, într-un ciclu care începe la 8 AM şi se sfârşeşte la 12AM. Obiectivul de ansamblu este să se asigure că toate programele "batch" sfârşesc execuția înainte de 8 AM în dimineața următoare (Fig. 5.8).

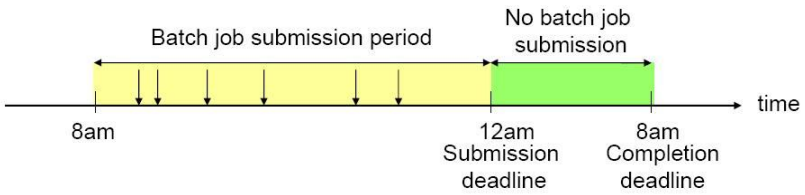


Fig. 5.8 Program batch

Scopul experimentului a urmărit verificarea posibilităților de a minimiza numărul de servere folosite în grupul de calcul, în condițiile îndeplinirii obiectivului de ansamblu definit mai sus. (Fig. 5.9)

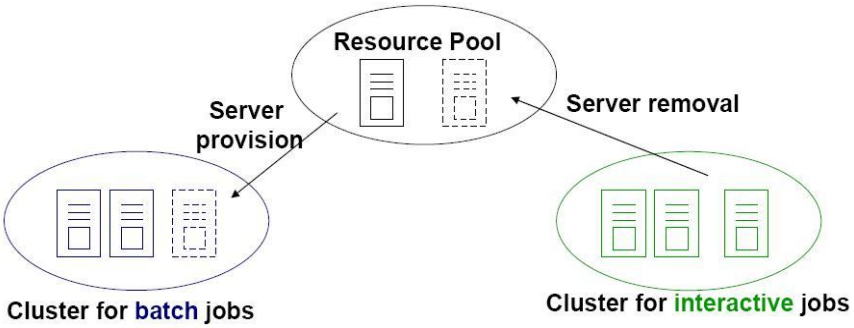


Fig. 5.9 Obiectivele serverelor din cluster

În Fig. 5.10 se prezintă o modalitate practică de implementare a unui analizor de obiectiv OA, într-o aplicație care include un sistem de tip grid.

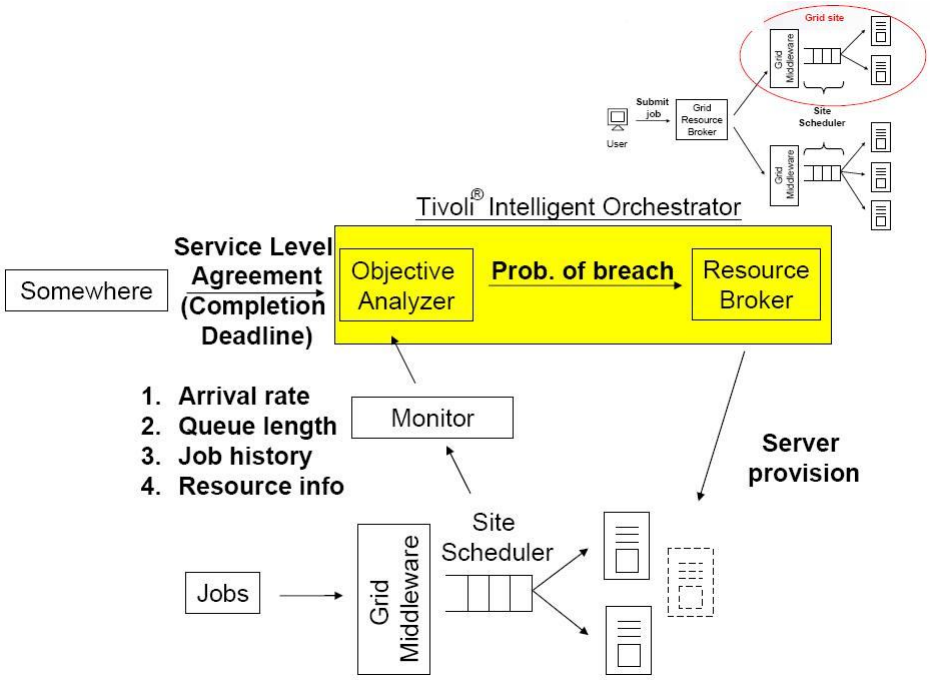


Fig. 5.10 Implementarea analizorului de obiectiv

Un analizor de obiectiv pentru un sistem grid se caracterizează prin:

- Supraveghează un grup divers de calculatoare, cu sisteme de operare diverse.
- Este capabil sa prezică rata de cereri http, lungimea cozii de asteptare, timpul de execuție pentru programele “batch”.
- Calculează probabilitatea de încălcare a obiectivelor stabilite, de exemplu numărul de programe care nu-şi termină execuția înainte de 8 AM.

Algoritmul folosit pentru estimare probabilitatii este următorul:

1. Se modelează rata de sosire a cererilor http din informația din loguri.
2. Se estimează CPU necesar pentru executarea programelor “batch”:
   a. pentru programe ce vor fi active în viitor
   b. pentru programe queue (în coada de aşteptare) pentru execuție
   c. se ajustează estimatul de CPU pentru diferite tipuri de calculator.
3. Se simulează comportamentul programului de alocare prin alocarea programelor la noduri (calculatoare) şi determinând ordinea de rulare.
4. Se determină, prin simularea valorii probabilității – procentul de programe care nu-şi termină execuția înainte de termenul stabilit.

Pentru experimentare, s-a utilizat schema descrisă în Fig. 5.11, cu o alocare de trei servere.

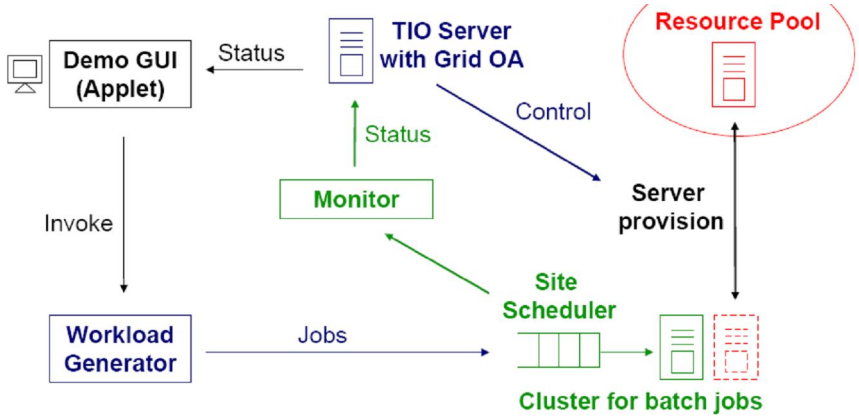


Fig. 5.11 Schema utilizată la experimentare

Rezultatele experimentale sunt prezentate sintetic în Fig. 5.12

S-a verificat de asemenea(Fig. 5.13) impactul folosirii unui controler ca TIO asupra performanței unui sistem de tip grid. Aşa cum era de aşteptat, performanța de ansamblu a sistemului se degradează în condițiile în care TIO nu este în situația să adauge un server atunci când se determină nevoia de resurse de calcul suplimentare.

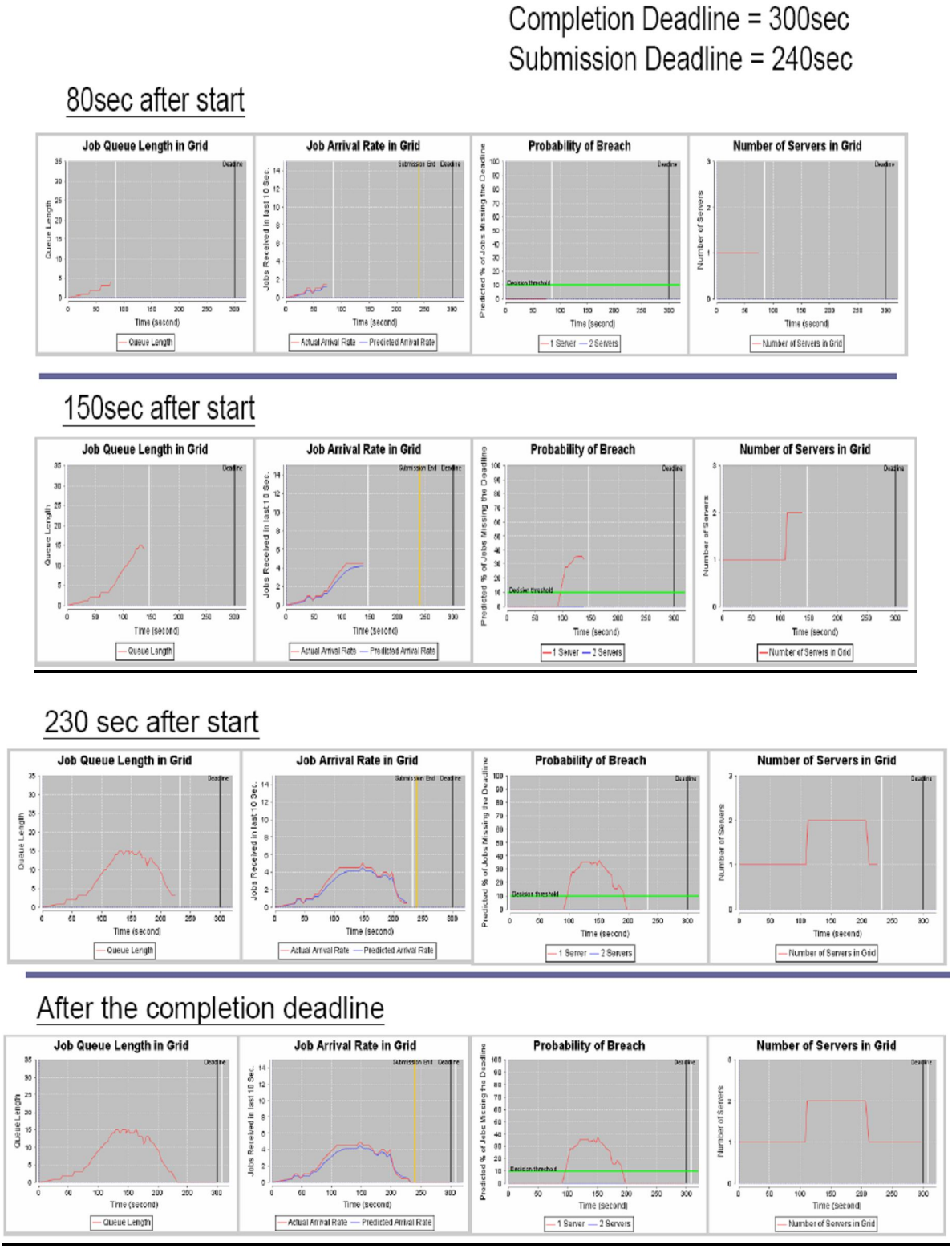


Fig. 5.12 Rezultate experimentale cu sistemul grid

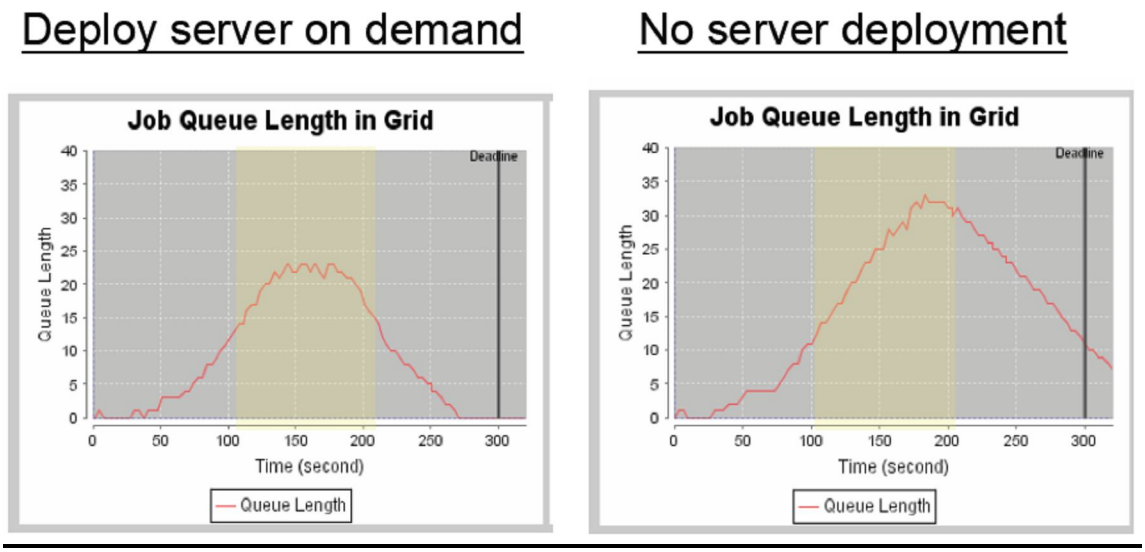


Fig. 5.13 Performanțele sistemului cu/fără controler TIO

5.3 SIMULAREA CONTROLULUI AUTOMAT CU DOMINO SERVER

În Fig. 5.14 este reprezentată schema simplificată a unui ansamblu de test pentru aplicațiile de tip Domino Server folosită în serviciul de poştă electronică la IBM [DI-07]. Monitorizarea sistemului se face prin citirea şi interpretarea logului. Controlerul este simulat ca un integrator. Acest experiment demonstrează posibilitatea folosirii teoriei controlului automat pentru administrarea sistemelor de calcul complex, principalele rezultate experimentale fiind prezentate în Fig. 5.15.

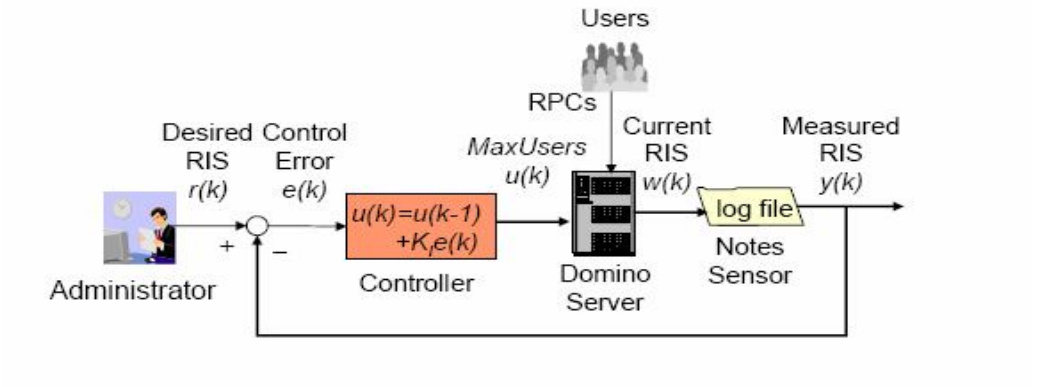


Fig. 5.14 Aplicație tip Domino Server

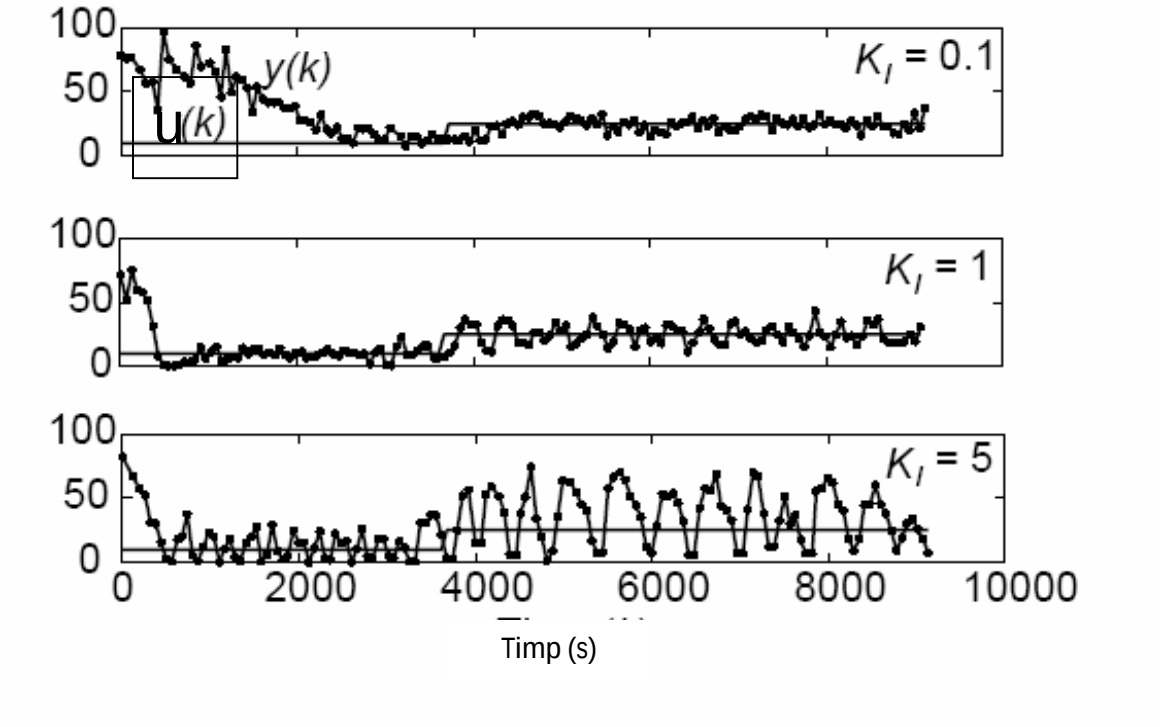


Fig. 5.15 Rezultate experimentale

Pentru simulare s-a folosit un program simplu de tip "spreadsheet". Programul de simulare îndeplineşte următoarele cerințe:

a) Specificarea ecuației cu diferențe finite;

b) Observarea efectelor schimbărilor în carecteristicile sistemului ca de exemplu K1 şi modelul Domino Server;

c) Reprezentarea variabilelor în funcție de timp.

În Fig. 5.16, coloanele de la A la F corespund variabilelelor specificate în Fig. 5.14. Pe monitorul calculatorului se afişează o foaie de calcul tabelar ce descrie o simulare a sistemului din Fig. 5.14. Calculele dintr-o celula corespund unei ecuații cu diferențe finite pentru componentele din Fig. 5.14. De exemplu, celula D6 este suma dintre D5 si C6 şi timpul G5 din moment ce coloana D corespunde controlerului care are ecuația cu diferențe finite u(k) = u(k) + Kie(k). Graficul afişează valorile lui y(k) în funcție de timp. Astfel, dacă schimbăm caracteristica sistemului KI, modificând celula G5, vedem imediat efectul de convergență al lui y(k).

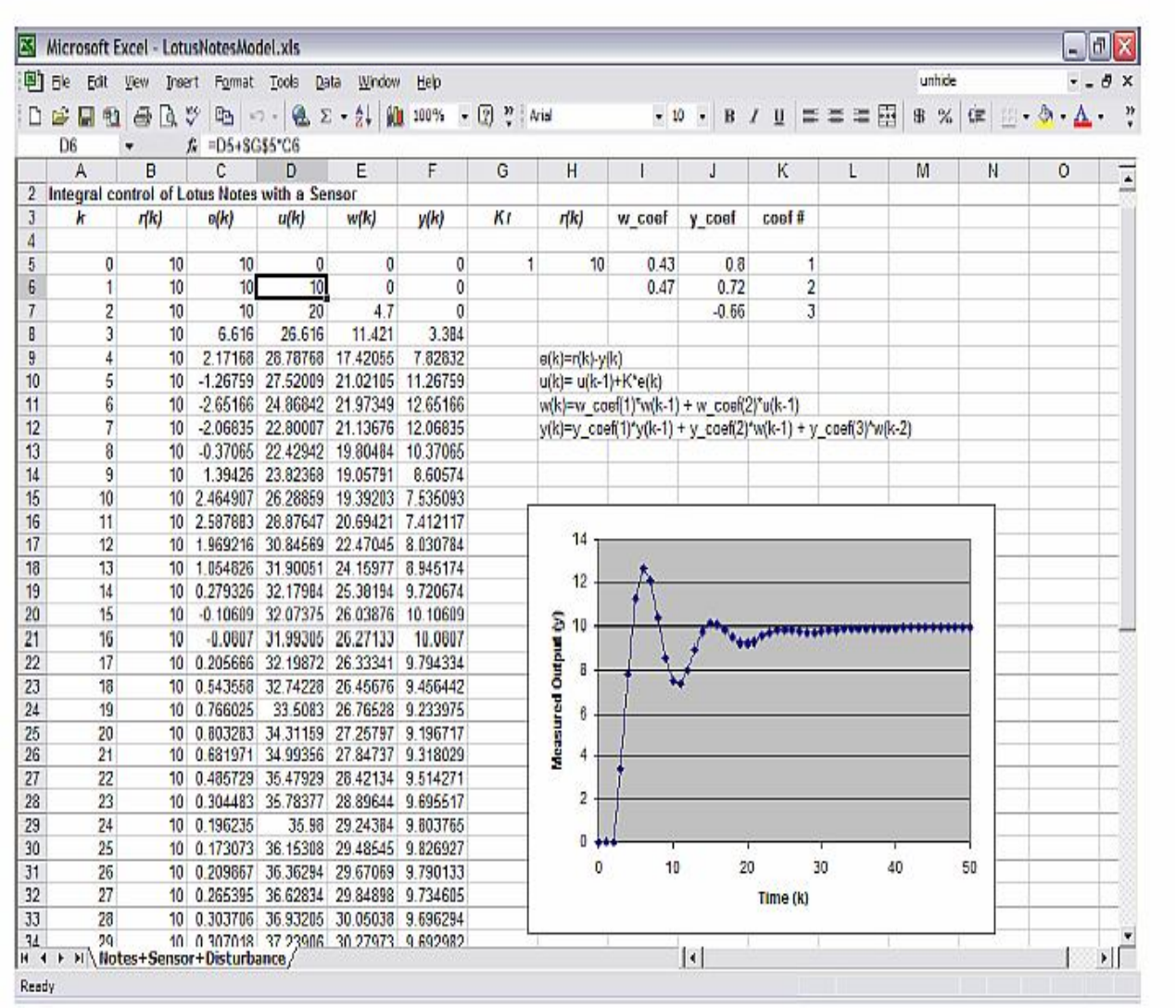


Fig. 5.16 Rezultatele simulării pentru sistemul Domino Server

5.4 MODEL COMUN TIO ŞI WEBSPHERE XD

În această aplicație, vom descrie două experimente, fiecare bazat pe ceea ce anticipăm că va fi un model de întrebuintare comun pentru TIO Global Resource Manager (de acum înainte indicat TIO-GRM) şi Websphere Extended Deployment (de acum înainte indicat WXD). În ambele experiente s-a demonstrat o colaborare eficientă între aceşti agenti care suportă obiective la nivel de serviciu specificat de administratori chiar şi în cazul în care se schimbă în mod dinamic spațiile de lucru. Pentru început, prezentăm un montaj comun (Fig. 5.17) utilizat în ambele experimente.

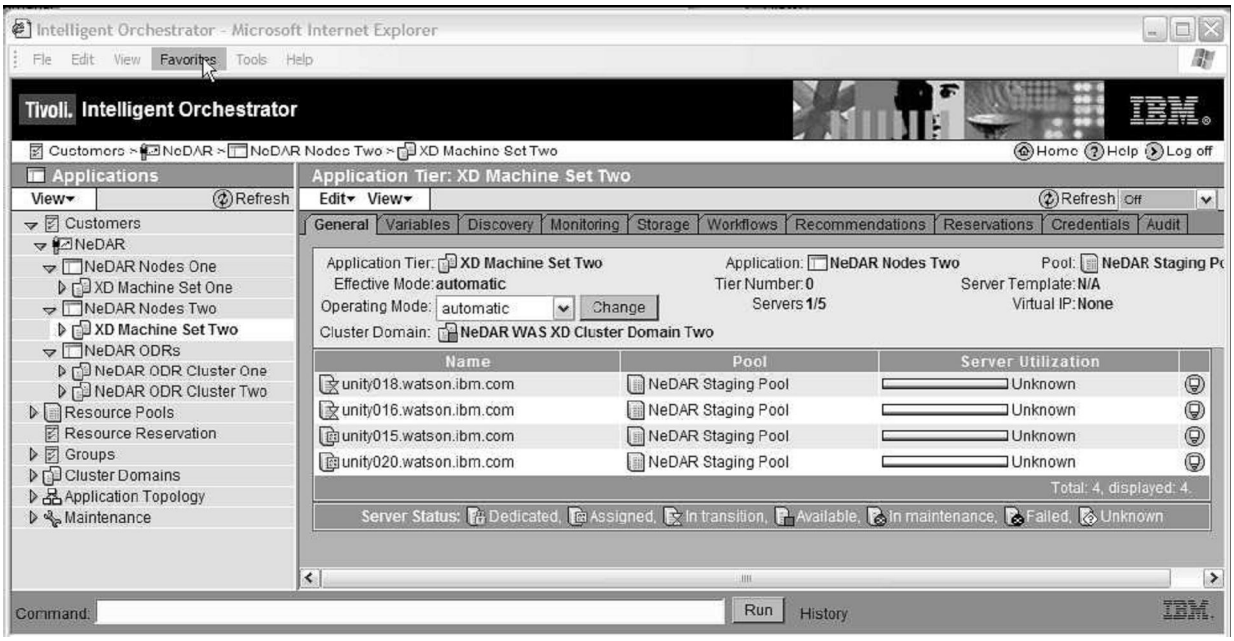


Fig. 5.17 : Screenshot de la Tivoli Intelligent Orchestrator.

- **Setarea experimentală**

Cele două experimente se bazează pe aceeaşi setare generală. În ambele cazuri, o instalare de TIO-GRM a fost configurată să conducă două instanțe independente a WXD, fiecare rulând pe propria lui aplicație cu un spațiu de lucru variabil în timp condus de Node Group. În plus, TIO-GRM a fost configurat să controleze un grup de servere libere omogene de la care cele două WXD sunt alimentate.

Pentru fiecare instalare de WXD, Deployment Manager şi ODR, fiecare au fost instalate pe servere separate dedicate (ex: patru servere s-au întrebuintat în total pentru conducerea WXD). Fiecare Deployment Manager a fost răspunzătoare de un singur Nod care rulează o singură aplicație instalată pe un singur Cluster Dinamic.

Am folosit aceeaşi aplicație bazată pe un server web pentru ambele instanțe de WXD. Fiecare aplicație constă dintr-o singură clasă de serviciu. Pentru fiecare clasă de serviciu, WXD permite unui administrator pentru a scrie o funcție de utilitate simplă a cărei valoare realizează un răspuns mediu de timp. Funcția de utilitate este descrisă printr-un format simplu: Administratorul impune un timp de răspuns la intrare RT0 şi selectează unul din şapte nivele de importanță – de la low la high. Aceşti parametri sunt introduşi într-o funcție de utilitate U(RT), în care utilitatea scade monoton crescător de la 1 la RT = la 0 a timpului de răspuns RT0. Pentru timpii de reacție mai mari decât RT0, utilitatea este negativă şi liniar descrescătoare cu o pantă dată de nivelul de administrare.

În cele patru experimente, ținta de răspuns a fost RT0 = 0.4 sec şi cu o pantă constantă pentru toți timpii de răspuns. Cererea în fiecare aplicație cuantificată de numărul de clienți expediând cereri la ODR, au fost generate folosind o singură buclă pentru un generator în bucla închisă. În principiu, fiecare cerere a necesitat acelaşi număr de cicli maşină. Timpul de « gândire » pentru fiecare client simulat este de 0.125 secunde.

Un total de şase servere omogene au fost disponibile pentru experimente. Fiecare server a fost un IBM eServer, o maşină din seria X335 cu două Intel Xeon 3.06GHz şi 2.5GB de RAM. Fiecare server poate servi până la 30-40 clienți.

TIO a fost instalat pe o maşină separată şi Global Resource Manager a fost configurat să aloce resursele la două WXD bazate pe informația de resursă U(n) furnizată de cele două Oas, la fiecare 30 secunde. Grupul de servere libere a fost plasat ca Uf= u_fn. În experiențele noastre, am setat uf = 0.5. De la discuțiile cu programatorii şi clienții TIO, este clar că există o bază solidă pentru folosirea grupului de server libere. Un server care stă în grupul de server libere poate fi mai ieftin decât unul WXD. Aplicațiile multi-server sau softul middleware țin seama de numărul de servere pe care este instalat. O maşină în aşteptare costă mai puțini bani în ceea ce priveşte rularea. Deşi este mai dificil a cuantifica, un server inactiv este avantajos deoarece el poate să fie angajat mult mai repede decât unul care este angajat în altă parte. Astfel în multe cazuri un administrator de centru de calcul doreşte să asocieze o valoare utilă la dimensiunea grupului de servere libere.

TIO-GRM a garantat cel puțin un server care să fie disponibil la fiecare instanțe de WXD. Astfel, fiecare WXD a rivalizat în esență cu altă instanță de WXD şi grupul de servere libere pentru serverele rămase. Scopul TIO-GRM a fost să găsească alocarea care a maximizat utilitatea sistemului, care a fost suma utilităților provenite din cele două WXD.

În cele patru experiemente ale noastre, TIO-GRM a fost setat de pe modul manual pe modul automatic.

- **Experimentul 1**

Rezultă din aplicația în care TIO-GRM controlează cele două instanțe WXD notate XD1 şi WXD2, reprezentate şi în Fig. 5.18. În figură avem o reprezentare pentru o perioadă de 8000 secunde (2 ore şi 20 minutele). De jos în sus avem:

A) numărul de clienți care solicită serviciul în aplicația notată WXD1.

B) numărul de servere programate să fie alocate(dorite) şi numărul de servere alocate lui WXD1.

C) timpul de răspuns mediu realizat cu WXD1.

D) utilitatea U(n) raportată de WXD1 la TIO-GRM

E) numărul de clienți cerând serviciul instalat pe WXD2.

F) numărul de servere programate să fie alocate (dorite) şi numărul de servere alocate lui WXD2.

G) timpul de răspuns mediu realizat cu WXD2.

H) utilitatea U(n) raporta de WXD2 la TIO-GRM şi

I) utilitatea totală reală obținută cu TIO-GRM de la WXD1, WXD2 şi grupul de servere libere FP (Free Pool). Timpii sunt marcați de etichetele din partea de jos a figurii.

La timpul 0, fiecare WXD a avut un server alocat şi numărul de clienți pentru fiecare cerere a fost zero. Deci toate utilitățile raportate de WXD1 şi WXD2 au fost inițial zero.

La momentul de timp a numărului de clienți care rulează pe WXD1 a sărit la 15. WXD1 deja a avut un server în Node Group şi după o perioadă tranzitorie, timpul de răspuns a fost de 0.23sec. Utilitatea reală relatată cu WXD1 la TIO a fost 0.44 sec. WXD1 estimează că nu este nici o creştere la utilitatea U(n).

De la timpul a la b, numărul de clienți în aplicația WXD1 a crescut. Este posibil de a asocia o creştere a timpului de reacție şi o scădere a valorilor de utilitate. La timpul b, când numărul de clienți a crescut la 50, timpul de răspuns mediu este 0.4 sec. Acest fapt a dus la scăderea utilității WXD1 şi cum U(1) mai mic decât U(2), TIO-GRM a început procesul de alocare a unui server adițional la WXD1. Alocarea necesită copierea softului necesar infrastructurii cererii şi poate dura 5 minute. Oricum, odată ce serverul este pregătit pentru WXD1 la timpul c, este o scădere a timpului de răspuns şi creşterea utilității lui WXD1 .Odată ce serverul a fost alocat la WXD1, utilitatea grupului de servere libere scade la 0.15.

Peste timpul c, numărul de clienți asociați cu WXD1 continuă să urce la 90. Ca răspuns la scăderea suplimentară în utilitatea relatată, TIO-GRM hotărăşte să aloce al treilea server WXD1 la timpul e.

La timpul g, tendința de creştere a numărului de clienți este inversată cu numărul clienților scăzând sub 30. Toate valorile utile estimate în creşterea WXD1 determină TIO-GRM să ia un server de la WXD1. În acest caz, odata ce serverul este întors la grupul de servere libere, este în stare să furnizeze un timp de răspuns de 0.4 sec.

În timp ce TIO-GRM şi WXD1 au fost implicați în acest schimb între servere, notăm că la timpul d, WXD2 a luat nişte clienți pentru prima dată şi numărul de clienți a început să crească. TIO-GRM a oferit un server adițional lui WXD2. Alocarea unui server la WXD2 şi dealocarea unui server de la WXD1 în timpul g, subliniind capacitatea TIO-GRM de a manipula procesele de alocare în timp.

La timpul h, am testat montajul provizoriu cu încărcare mare pe WXD1 si WXD2. Imediat după decizia TIO-GRM de a aloca ultimul server rămas memoriei libere a WXD1 la timpul h.

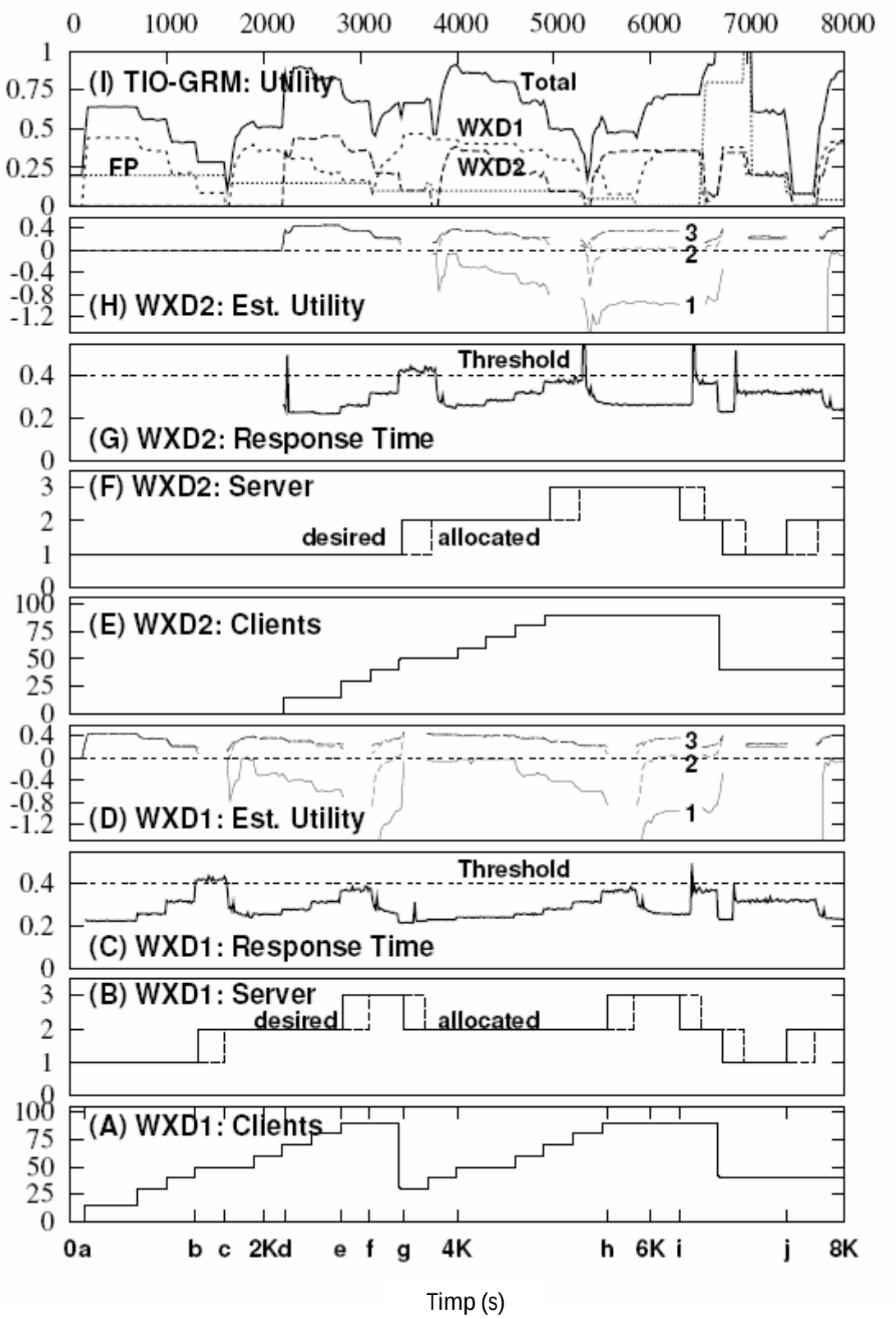


Fig. 5.18: Rezultatele experimentului 1 pentru nouă serii dinamice

Pentru a ilustra abilitatea sistemului de a răspunde dinamic la schimbările în obiective, şi pentru a demonstra utilitatea grupului de servere libere, utilitatea acestui grup este crescută de la 0.05 la 0.4. Cum am discutat mai sus, această reprezintă o schimbare în licența structurii. Fig. 5.18 demonstrează că TIO-GRM poate accepta această modificare fără a fi nevoit să dea restart. În acest caz, TIO-GRM a scos afară serverele ambianțelor WXD1 şi WXD2. În timp ce această acțiune cauzează timpii de răspuns a WXD1 şi WXD2 să crească, precum şi utilitățile lor.

Mai târziu, când încărcătura pe ambele WXD1 şi WXD2 a scăzut, o serie complementară de paşi este declanşată la timpul j, când administratorul de sistem determină o scădere a valorii grupului de servere libere pe fiecare server 0.01. Ca rezultat, TIO-GRM renunță la un server adițional din grupul de servere libere a fiecărui WXD.

Acest experiment demonstrează eficiența utilizării funcției de utilitate în implementarea strategiei generale de alocare de resurse.

- **Experimentul 2**

Am realizat în esență acelaşi montaj ca şi cel utilizat în Experimentul 1 pentru a demonstra eficacitatea şi siguranța agenților. Pentru a furniza o incărcătură variabilă în timp am configurat fiecare generator de încărcătură pentru a crea o emulație realistă a variației de timp a parametrilor. Numărul de clienți în bucla închisă a fost resetat din minut în minut.

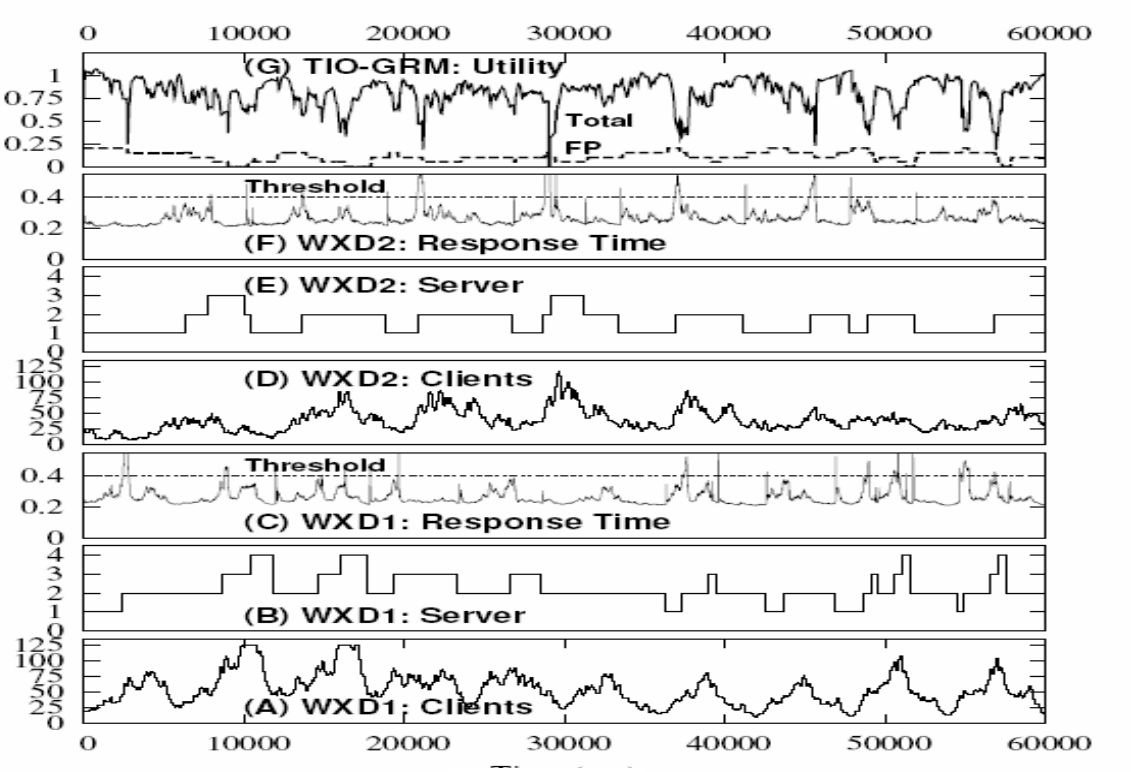


Fig. 5.19: Rezultatele experimentului 2 pentru sase serii dinamice.

Seriile de timp a spațiilor de lucru dinamice pe fiecare cerere au fost generate în mod independent de cererea XD1 primind condiții de încărcare mai mari decat XD2.

TIO-GRM a fost iarăşi setat pe modul automat şi experimentul a rulat neîntrerupt pentru o perioadă de peste 60000 secunde (16 ore şi 40 minute).

Rezultatele Experimentului 2 cu TIO-GRM, care administrează WXD1 şi WXD2, sunt prezentate în Fig. 5.19. Figura se identifică în şase serii de timp alocate dinamic. De jos în sus semnificația lor este următoarea:

A) numărul de clienți cerând serviciul în aplicația instalată pe WXD1.

B) numărul de servere programate să fie alocate pe WXD1.

C) timpul mediu de răspuns realizat cu WXD1.

D) numărul de clienți solicitând serviciul în aplicația instalată pe WXD2.

E) numărul de servere programate să fie alocate pe WXD2, i

F) timpul mediu de răspuns realizat cu WXD2.

G) sistemul şi utilitatea totală reală obținută cu TIO-GRM de la WXD1, WXD2, şi polul liber (FP).

Rezultatele arată ca peste 16 ore de curs, toti agenții au colaborat cu succes. Printr-un schimb armonios între servere, fiecare instanță de WXD a fost în stare să atingă pragul de sensibilitate al timpului de răspuns. Sistemele dinamice cu timp de întârziere mare adesea suferă de efecte neliniare ca de exemplu: histereza şi invariația în timp. Sistemul XD-TIO-GRM studiat are delay-uri mai mari de 5 minute - condiții minuscule care în anumite împrejurări pot să ducă la abateri de comportare.

Oricum, pentru spațiile de lucru ce variază în timp experimentate prin WXD1 si WXD2, alocarea rapidă repetată şi dealocarea de servere cu TIO-GRM nu au fost observate, în ciuda întârzierilor de alocare.

5.5 CONTROLUL AUTOMAT ÎN MEDIUL VIRTUAL

Tehnica aplicată în acest caz este bazată pe măsurarea valorilor în timp real a sarcinii generate de unitatea centrală pentru procesele în execuție pe fiecare server, calculându-se raportul dintre cele două. Pe baza coeficientului cunoscut dintre etapa de alocare a sarcinilor, coeficienții unității centrale de procesare sunt ajustați pe măsură ce serverele deservesc cererile individuale ale clienților [DI-08].

Deoarece realocarea coeficienților unității centrale de procesare este un proces lent şi ar trebui să reacționeze unor schimbări pe termen lung ale sarcinii, mecanismul de răspuns trebuie să elimine toate variațiile rapide ale parametrilor măsurați, de exemplu ale sarcinii unității centrale de procesare. Ca urmare, ieşirea traductorului este condiționată, calibrată şi introdusă într-un filtru de mediere.

În acest experiment arhitectura sistemelor de calcul autonome ia în considerare faptul că sistemul de calcul a cărui ieşire este sarcina unității centrale de procesare este neliniar, stohastic şi cu parametri lenți. Arhitectura componentelor pentru un controler adaptiv robust pentru un serverul virtual al unui sistem de calcul autonom este dată în Fig. 5.20 de mai jos. Urmărind rezultatele teoretice obținute în [9], se prezintă, pentru cazul timpului discret, un controler adaptiv robust direct pentru modelul sistemul de calcul autonom considerat ca fiind un sistem neliniar incert având o incertitudine parametrizată, constantă şi neliniară şi o incertitudine neliniară dependentă de stare. Rezultatele obținute în [9] garantează că, controlerul propus este stabil Lyapunov. Acest lucru garantează o stabilitate parțială robustă asimptotică a sistemului în buclă închisă.

Altfel spus, sistemul de calcul autonom în buclă închisă este stabil, asimptotic şi robust la stările asociate cu coeficienții de alocare a unitătii centrale de procesare.

Modelul sistemului considerat este descris de un sistem de timp discret neliniar având atât incertitudini parametrice cât şi neparametrice. Pentru toate serverele funcționând pe acelaşi suport hardware controlerul adaptiv robust calculează procentajul de alocare al unității centrale de procesare.

Considerând că pe fiecare suport hardware al fiecărui calculator rulează j servere virtuale diferite, atunci sistemul de calcul autonom va rula j controlere adaptive robuste independente. Cu toate acestea, aceasta nu reprezintă o soluție realistă deoarece implică consumarea unei bune părți a puterii de calcul pentru controlul sarcinii.

Prin urmare resursele existente ale calculatorului vor fi alocate proceselor active. Se va considera că pentru fiecare hardware distinct se va calcula la intervale de timp depinzând de dinamica lentă a serverului ce guvernează procesul o singură lege adaptivă robustă.

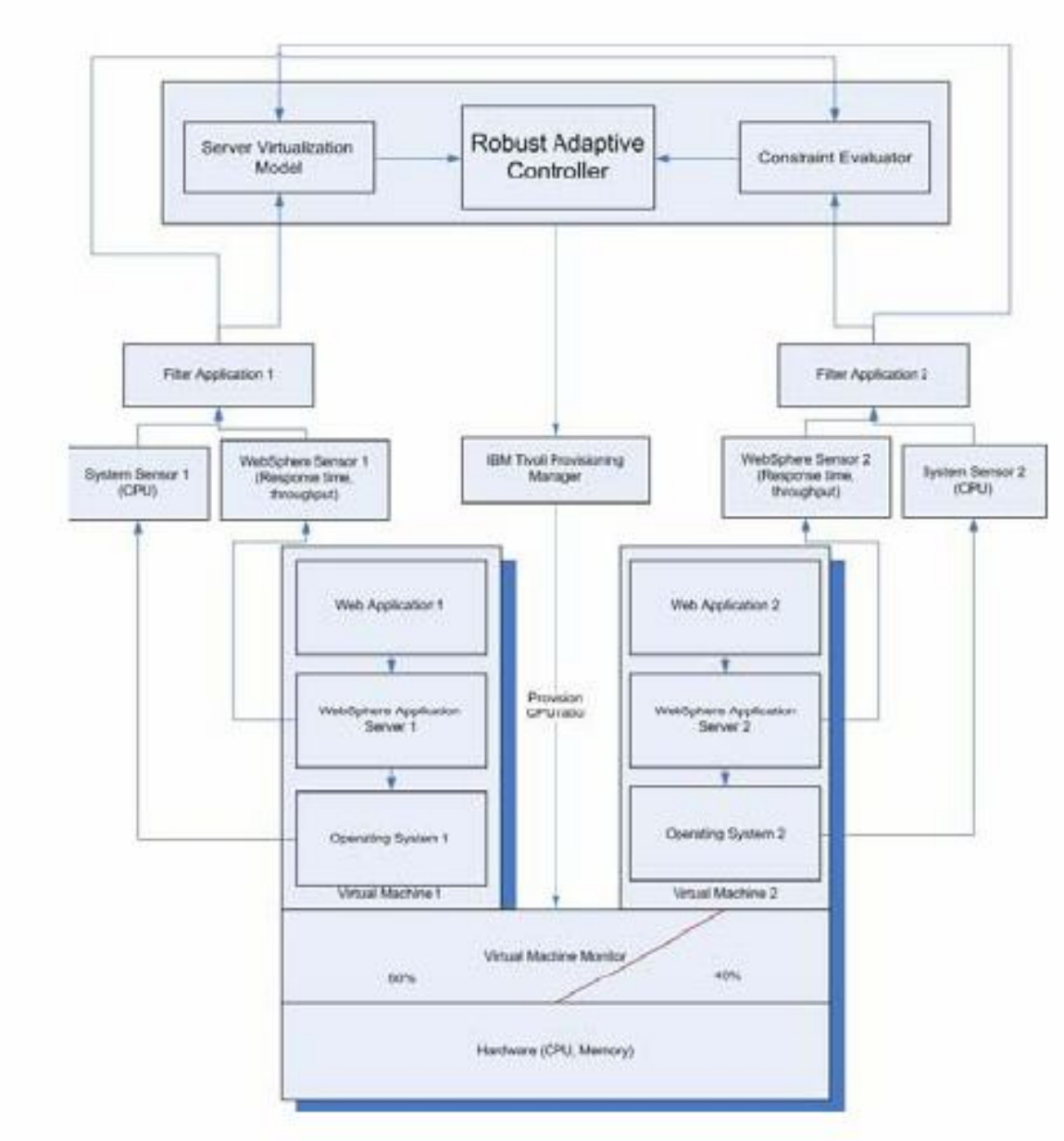


Fig. 5.20 Arhitectura la nivel de componente pentru un sistem de calcul autonom adaptiv robust utilizat pentru controlul sarcinii unității centrale de procesare a unui mediu cu servere virtual

- **Implementarea**

Arhitectura prezentată mai sus (Fig. 5.20) pentru autoconfigurarea sarcinii unitatății centrale de procesare utilizând un controler adaptiv robust a fost implementată şi testată pe o rețea de calculatoare cu funcționare în timp real. Configurația de testare este prezentată în Fig. 5.21.

Serverul virtual a fost pus în funcțiune prin instalarea unui program VMM pe un calculator dotat cu procesor Intel Core 2 Duo.

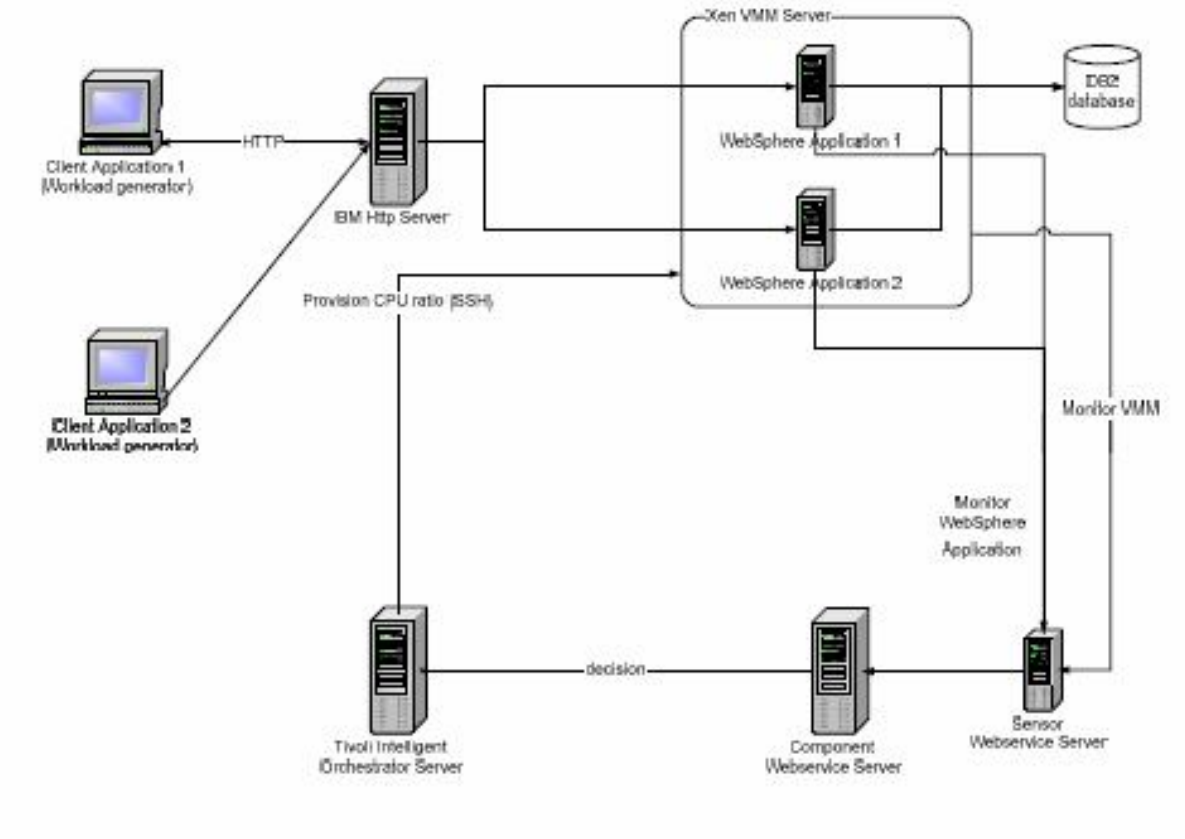


Fig. 5.21 Configurația de testare utilizată

Actuatorul întregului sistem, reprezentat de IBM Tivoli Provisioning Manager (TPM) a fost instalat pe un alt calculator IBM având un procesor ce funcționează la 2.6 GHz. Calculatoarele virtuale s-au obținut prin paravirtualizare cu Debian Etch Linux. Componentele mediului de calcul autonom au fost implementate ca servicii web, toate rulând pe un calculator dotat cu processor Intel de 3.2 GHz. Pentru configurarea întregului sistem a fost proiectată o interfață utilizator grafică.

Pentru implementare s-au folosit biblioteci JavaServer Faces, după cum este descris în [7]. Controlerul adaptiv robust a fost inclus în lista de controlere a interfeței, aşa cum se exemplifică în Fig. 5.22.

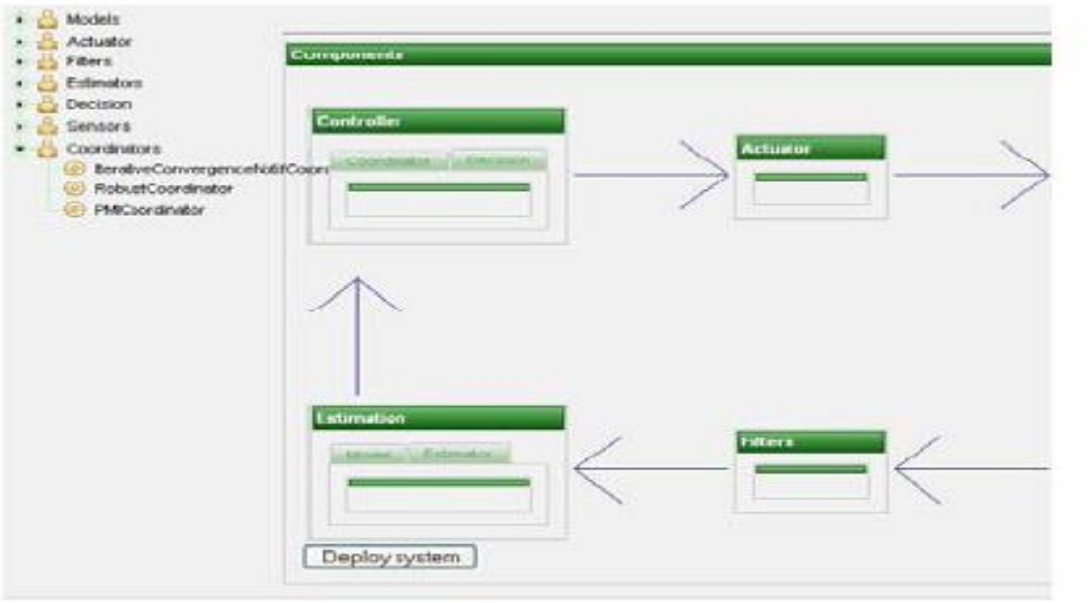


Fig. 5.22 Interfața grafică cu utlizatorul care permite configurarea tipului de controler utlizat în sistemul de calcul autonom

Controlerul este selectabil dintr-o listă de algoritmi sintetici de control. Până în acest moment sunt disponibile câteva controllere: ConvergenceNotificationController, RobustAdaptiveController şi PMController, care este un controler clasic proporțional. Pentru arhitecturi simple de servere virtuale ce nu necesită instalarea unui hardware şi a unui software complex, se poate utiliza o lege de control simplă, care oferă rezultate bune pentru controlul ceoficienților unității centrale de procesare. Algoritmi mai sofisticați se pot utiliza pentru arhitecturi complexe de servere, în sisteme critice de operare.

Instrumentul de proiectare dezvoltat, prezentat în Fig. 5.22, permite configurarea mai multor modele de control. Pentru configurarea sistemului, inginerul de sistem selectează cu mouse-ul componente, în timp ce componetele sistemului pot fi conectate şi pornite în timpul funcționării. Utilizatorul trebuie să conecteze componentele prin săgeți. Fig. 5.23 arată fereastra de configurare a stării inițiale a senzorilor.

Sarcina unității centrale de procesare, timpul de răspuns şi numărul de apeluri sunt setate pentru a furniza coordonatorului valorile coresounzătoare măsurate. Filtrul este configurat tot printr-o fereastră de configurare care este prezentată în Fig. 5.24.

Estimatorul este configurat folosind fereastrele din Fig. 5.26 şi 5.27. Configuararea estimatorului include alegerea unui model, fapt ilustrat în Figura 5.25 de mai jos, unde este prezentat modul în care variabilele filtrate sunt transformate în variabilel modelate a variabilelor filtrate. Parametrii care trebuie setați sunt numarul de iterații necesare pentru obținerea unei estimări cât mai bune pentru variabilele modelate, măsura convergenței şi perioada de eşantionare, aşa cum se explică în această parte.

Un exemplu pentru această configurare este dat în Figura 5.27. Blocul decizional implementează legea adaptivă robustă elaborată în Capitolul 3.

Fig. 5.23 Fereastra de configurare a senzorului

Fig. 5.24 Fereastra de configurare a filtrului

Legea de control este interpretată şi se ia decizia modificării sarcinii unității centrale de procesare funcție de anumite praguri care sunt utlizate pentru a atenua variațiile rapide ale sarcinii unității centrale de procesare. Controlerul trimite comenzile adecvate actuatorului reprezentat de IBM Tivoli Provisioning Manager.Pe baza valorilor obținute în urma comparării controlului realizat de controlerul robust adaptiv şi factorii de atenuare reprezentați sub formă de praguri, sistemul furnizează actuatorului modul în care să acționeze. Inginerul de sistem trebuie să îi indice sistemului care actuator este utilizat şi pe care calculator fizic se află acesta, astfel încât să poată lansa toate comenzile legate de configurarea întregului mediu de calcul autonom şi să specifice care parametru trebuie utilizat pentru inițializarea actuatorului.

Configurarea legăturilor dintre controllerul adaptiv robust şi IBM Tivoli Provisioning Manager se face prin intermediul unei ferestre de configurare, aşa cum se vede în Figura 5.29.

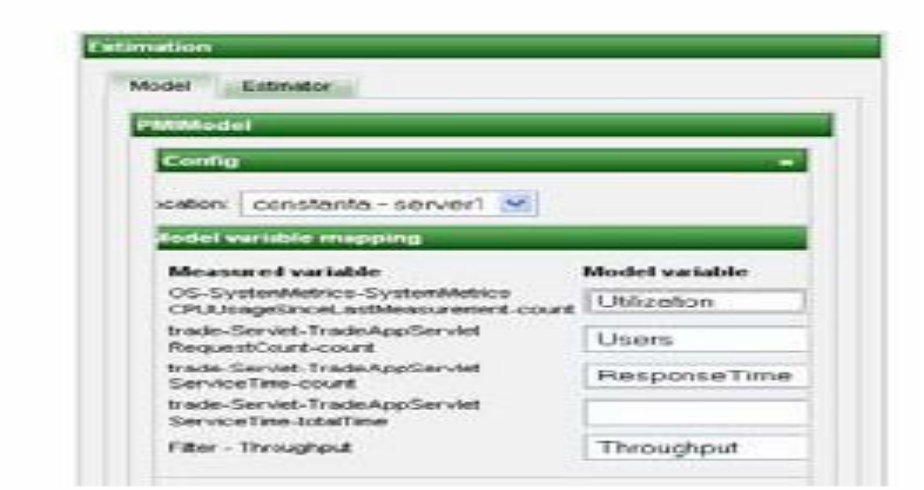


Fig. 5.25 Fereastra de configurare care permite utlizatorului alegerea unui model potrivit

Fig. 5.26 Fereastra de configurare a timpului de procesare şi a utlizării estimatorului

Fig. 5.27 Fereastra de configurare a parametrilor estimatorului

Fig. 5.28 Fereastra de configurare a parametrilor blocului decizional

Fig. 5.29 Fereastra de configurare a serverelor actuatorilor

5.6 CONCLUZII ŞI CONTRIBUȚII

Capitolul 5 al tezei de doctorat intitulat "Rezultate experimentale" a urmărit elaborarea unor sisteme de testare flexibile, capabile să permită verificarea caracteristicilor stabilite în dezvoltările teoretice din Capitolele 1, 2, 3 şi 4. Structurile sistemelor autoadministrate propuse pentru experimentare au fost dezvoltate în laboratoarele de cercetare ale IBM din Canada şi SUA.

Din analiza rezultatelor experimentale se desprind următoarele concluzii:

- Arhitectura sistemului autoadministrat propus în Aplicația 1 se bazează pe structurile autonome şi reconfigurabile dezvoltate în Capitolul 2 şi are la bază principiile sistemelor autotestabile tolerante la defecțiuni prezentate în Capitolul 1. Structura hardware este minimală(doi controleri), iar implementarea software este specifică necesităților comerțului electronic distribuit – WebSphere Application Server, furnizat de IBM. Au fost analizate principalele mecanisme care stau la baza dezvoltării sistemelor autonome, bazate pe reconfigurare, atât pe durata timpului de rulare a programului, cât şi prin adăugarea unor bucle suplimentare de control automat. Graficele prezentate în Fig. 5.2-5.6 analizează distribuția temporală a utilizatorilor, timpul de răspuns sau efectul funcționării serverelor din cluster. Testul efectuat permite evaluarea numărului optim de controlere care participă la analiza parametrilor unei aplicații concrete.

- Aplicația 2 prezintă un sistem Grid autonom care permite adăugarea automată de servere în scopul administrării optimale a resurselor disponibile într-un centru de calcul. La sistemul TIO disponibil a fost conectat un Analizor de Obiectiv care îndeplineşte funcțiile de monitorizare/supraveghere a resurselor pentru a identifica deciziile de utilizare ale cererilor http şi de a estima şi raporta probabilitățile de îndeplinire a obiectivelor de serviciu. A fost implementat un algoritm de predicție (descris în Capitolul 4) pentru estimarea probabilităților de apariție a erorilor într-un cluster format din trei servere. Analiza rezultatelor experimentale prezentate în Fig 5.12, Fig 5.13 am demonstrat că performanțele globale ale sistemului se degradează în situația în care TIO nu poate adăuga un nou server atunci când se solicită resurse de calcul suplimentare.

- A fost imaginat un experiment în Aplicația 3, care demonstrează importanta utilizării teoriei controlului automat în administrarea sistemelor de calcul complexe. Ansamblul de test prezentat în Fig. 5.14 este o aplicație de simularea Controlului automat pentru un Domino-Server folosit de poşta electronică la IBM. Pentru simulare s-a folosit un program "Spreadsheet" care rezolvă ecuații cu diferențe finite şi reprezintă grafic variabile în funcție de timp (Fig. 5.15 şi Fig. 5.16).

- În Aplicația 4 am efectuat două experimente în care am utilizat un model comun TIO-GRM şi WXD simulând aceleaşi obiective la nivel de serviciu, dar care se schimbă aleator în spațiul de lucru. Analizând rezultatele experimentale oținute pentru cele două experimente (Fig. 5.18 respectiv Fig. 5.19), se constată printr-o comutare optimală a serverelor se poate atinge pragul de sensibilitate propus şi timpul de răspuns programat. Se mai poate constata că pentru sistemele dinamice cu timp de întârziere mare apar efecte neliniare ca de exemplu: histereza şi invarianta în timp. Pentru spațiile de lucru cu viteză scăzută de variație în timp, alocarea sau izolarea repetată de servere cu TIO-GRM nu este sesizabilă.

- Ultima aplicație propusă pentru experimentare "Controlul Automat în mediu virtual" este o simulare a performanțelor obtenabile utilizând metoda şi algoritmul de control adaptiv robust, prezentată în Capitolul 3. În acest experiment, s-a propus o arhitectură originală pentru sistemul autonom de calcul a cărui sarcină activă este o unitate centrală de procesare cu parametrii lent variabili în timp. Pentru a obține facilități sporite la simulare, s-a presupus că la fiecare suport hardware al fiecărui calculator conectat la cluster, rulează servere virtuale diferite a căror funcționare este gestionată de sistemul de calcul autonom, adaptiv robust.

Intefața grafică cu utilizatorul (Fig. 5.22) permite configurarea tipului de controler dorit (CNC, RAC sau PMC) pentru conectarea la sistemul de calcul autonom. Fig. 5.24, 5.29 prezintă ferestrele de configurare a parametrilor de procesare/analiză/decizie pentru diferite configurații de servere (senzor, estimator, filtru, actuator).

Principalele contribuții originale aplicative aduse la elaborarea Capitolului "Rezultate experimentale" pot fi sistematizate astfel:

- Conceperea unei arhitecturi optimale(complexitate/preț de cost) de sistem autoadministrat, capabil să se adapteze testelor de mentenanță utilizând programe dedicate disponibilă la IBM.

- Asocierea noțiunii de client cu un generator de trafic virtual capabil să modifice rata de emisie între o distribuție liniară şi una Gaussiana pe baza unei cereri http URL pentru o adresă specificată.

- Implementarea buclei de calcul autonom TIO la controlul şi ajustarea dinamică a numărului optim de calculatoare participant la aplicație.

- Programarea buclei de control pentru a estima efectul adăugării/izolării unui server în diferite configurații de operare.

- O analiză comparativă a performanțelor care se obțin prin utilizarea unui controler cu prag (simplu şi ieftin) sau a unui controler complicat cu reacție feed-forward (complicat cu timp mare de autoreglare).

- Experimentele au arătat că este posibil să se selecteze o varietate bogată de modele de control dependente de distribuția cererilor de utilizare.

- S-a demonstrat experimental că nu este avantajos să se păstreze acelaşi tip de controler pe toată durata rulării programului de procesare, atunci când distribuția cererilor de utilizare este variabilă.
- Pentru automatizarea procesului de adăugare/înlăturare a unui server din cluster pe baza calculării obiectivelor de performanță; elementul de control TIO a fost completat cu un modul adițional denumit Analizor de Obiectiv (AO).
- Elaborarea unei arhitecturi experimentale de tip grid capabilă să funcționeza atât cu sarcini interactive cât şi pentru încărcări de tip batch.
- Implementarea software a unui algoritm de testare pantru estimarea probabilității de apariție a erorilor de încălcare a obiectivelor în sistemele grid prevăzute cu AO.
- Măsurări experimentale privind performanțele sistemului cu/fără TIO în condiții de test diferite, interpretarea rezultatelor şi concluziile ce se impun.
- Conceperea unui ansamblu de test simplificat pentru o aplicație de tip Domino Server folosită de poşta electronică laIBM, utilizând teoria controlului automat în monitorizarea sistemelor complexe de calcul.
- Simularea controlului automat pentru un Domino Server utiliând programul spreadsheet care rezolvă ecuații cu diferențe finite şi afişează rezultatele funcție de timp.
- Elaborarea unui model comun de analiză pentru TIO Global Resource Manager şi WebSphere Extended Deployment care simulează funcționarea în condițiile în care spațiile de lucru se schimbă dinamic.
- S-a demonstrat experimental că pentru spațiile de lucru variabile în timp alocarea/dealocarea rapidă de server la cluster, nu este sesizabilă, datorită controlului precis al întârzierilor de alocare.
- Elaborarea unei arhitecturi experimentale la nivel de componente pentru un sistem autonom de calcul adaptiv robust, utilizat pentru controlul activ al sarcinii unității centrale de procesare într-un mediu cu servere virtuale.
- Simularea unui server virtual cu ajutorul programului VMM instalat pe un calculator performant dotat cu procesor Intel Core 2 Duo.
- Proiectarea şi implementarea software a unei interfețe grafice cu utilizatorul care configurează sistemul de test utilizând biblioteca Java Server Faces.
- Calculatoarele virtuale, actuatorii, senzorii, filtrul, estimatorii şi blocul decizional, au fost simulate prin paravirtualizare utilizând sistemul de operare Debian Etch Linux disponibil la IBM.

În final, putem concluziona că cele cinci structuri de sisteme autoadministrate propuse pentru testele experimentale de verificare şi diagnoză, au furnizat informații relevante care consolidează suportul matematic al dezvoltărilor teoretice din teza de doctorat.

6 CONTRIBUTII ORIGINALE, CONCLUZII FINALE SI CERCETARI DE PERSPECTIVA

Teza de doctorat prezentată a putut fi elaborată pe structura preocupărilor de cercetare ştiinţfică desfăşurate de autor în ultimii zece ani, în domeniul sistemelor digitale de calcul auto-administrate, auto-adaptabile şi auto-configurabile, tolerante la defectiuni. În final, vom prezenta cele mai semnificative contribuţii originale teoretice şi experimentale aduse la elaborarea tezei, concluziile relevante care se desprind şi vom jalona câteva dintre cele mai importante direcţii viitoare de cercetare ştiinţifică.

6.1 CONTRIBUŢII ORIGINALE

Pe parcursul redactării tezei de doctorat, la sfârşitul fiecărui capitol, am prezentat contribuţiile originale care reprezintă suportul dezvoltărilor teoretice şi experimentale din lucrare.

În sinteză, considerăm ca relevante la structura şi esenţa tezei de doctorat, următoarele contribuţii originale împărţite în două categorii:

- **Contribuţii teoretice**

- Analiza şi prezentarea în sinteză a stadiului actual al cercetărilor din domeniul sistemelor digitale autonome dinamice, tolerante la defecţiuni, în scopul identificării direcţiilor viitoare de cercetare.

- Prezentarea în formă compactă şi modelarea conceptului de toleranţă la defecţiuni în echipamentele electronice digitale cu structură redundantă protectivă la defecţiuni.

- Elaborarea unui model matematic pentru analiza toleranței la defecțiuni şi stabilirea diagnozabilității echipamentelor electronice de mare utilitate practică.
- Dezvoltarea algoritmilor de reconfigurare a sistemelor şi de mascare a defectelor în echipamentele numerice, prevăzute cu structuri redundante protective de tip static sau dinamic.
- Analiza structurilor redundante protective statice de tip individual sau global, rezultate prin procedee de multiplicare, a permis partiționarea optimală a schemei de bază în unități independente ce pot fi reparate individual.
- Evaluarea întârzierilor în propagarea informației prin modulele sistemului redundant şi limitările performanțelor dinamice pe care le presupun.
- Se propune o clasificare a sistemelor autonome în structuri redundante de tip static, dinamic sau distribuit, ca instrument eficace de elaborare optimală a topologiei magistralelor de date sau adrese tolerante la defecțiuni.
- Analiza detaliată a structurii redundantă dinamic, aplicată magistralelor de date a dovedit că reprezintă cea mai comodă metodă de implementare a toleranței la defecțiuni, comparativ cu alte metode cunoscute, care degradează parametrii dinamici sau limitează capacitatea de transmitere a informației.
- Sistematizarea principalelor criterii de comparare a performanțelor de fiabilitate şi definirea unor criterii noi (indicele de cost şi de eficiență) permite stabilirea celor mai potrivite metode de implementare a toleranței la defecțiuni pentru o aplicație practică bine definită.
- Am elaborat o metodă analitică pentru stabilirea nivelului optim de partiționări a unui sistem digital în vederea aplicării redundanței protective la defecțiuni.
- Elaborarea unei teorii unitare şi adaptarea conceptului de „sisteme autonome de calcul" ca bază a managementului optimal al procesului de automatizare a infrastructurii informaționale actuale.
- Elaborarea unei structuri generice pentru sistemele computerizate autonome, care reprezintă fundamentul utilizării teoriei de control automat, la structurarea conducerii echipamentelor numerice multiple, conectate la aceaşi unitate centrală de prelucrare.
- Analiza comparativă a diferitelor arhitecturi de sisteme numerice autonome cunoscute în literatura de specialitate, în vederea stabilirii soluției optimale, capabilă să se configureze pe structura existentă şi să se autoconfigureze pentru o mai bună gestionare a infrastructurii.
- Pornind de la constatarea că infrastructura IT a devenit vitală pentru funcționarea societăților şi chiar a guvernelor, şi că erorile de perturbare sau comunicare pot genera rezultate catastrofale, s-a lansat ideea construirii unui mediu software complex capabil să: supravegheze, optimizeze, protejeze şi să adapteze infrastructura IT existentă printr-o autogestionare inteligentă a platformei.

- Analiza caracteristicilor principale pe care trebuie să le indeplinească: Evaluatorul de constrângeri, Controlerul pentru adaptare, Executorul de adaptare ca şi celelalte blocuri funcționale a permis elaborarea unui „Banc de probă" care constituie suportul rezultatelor experimentale menite să confirme dezvoltările teoretice prezentate pe parcursul elaborării tezei de doctorat.
- Se demonstrează că, în prezent, tehnologiile comerciale ale MPC disponibile au la bază implementarea unui model liniar de control al sistemului. Procesele industriale reale cu variație rapidă în timp a parametrilor prezintă o neliniaritate a funcției de transfer nu poate fi ignorată. Analiza pe acească bază, conduce la scăderea performanțelor globale, datorită incapacității modelelor liniare de a aproxima cu precizie procesele neliniare.
- A fost efectuat un studiu al modelelor predictive de control neliniar în spațiul multidimensional bazate pe analiza în timp real a funcției de transfer pentru sistemele liniare discrete.
- S-a introdus conceptul de model predictiv de control cu neliniaritate izolata, care combină avantajele analizei proceselor de predicție liniară şi neliniară.
- În ipotezele de calcul specificate, problema optimizării sistemelor cu neliniaritate izolată poate fi rezolvată analitic şi nu necesită utilizarea algoritmilor numerici de optimizare care cresc costurile sistemelor de prelucrare.
- Elaborarea a două teoreme (mecanism stabil şi instabil) care stau la baza analizei stabilității nominale a sistemului proiectat.
- Analiza sistemelor de control adaptiv robust caracterizate prin semnale de intrare afectate de erori variabile şi invariabile. Se demonstrează că această soluție nu este optimală, deoarece implică un consum mare de resurse de calcul.
- Pentru a evita supra-parametrizarea modelelor de control adaptiv bazate pe identificator, s-a propus o constrângere suplimentară , care trebuie aplicată vectorului ce descrie legea controlului.
- Pe baza analizei efectuate asupra stabilității sistemelor de control autonom adaptiv s-a dedus empiric o condiție de inegalitate , pe care o satisfac erorile produse la prelucrare.
- Dezvoltarea suportului matematic pentru estimarea şi predicția zgomotelor, analizate ca şi procese aleatoare staționare.
- În funcție de momentele de timp la care se prelevează probe din semnalele am identificat trei tipuri generice de predicție: pe interval fix, cu punct fix şi cu avertizare.
- Evaluarea erorilor de modelare produse la predicție în ipoteza că intervalul de estimare este bine definit şi dispersia zgomotului prezent la intrare se situează într-o bandă limitată
- S-a demonstrat că dispersia erorii de estimare a procesului de predicție creşte nelimitat odată cu creşterea diferenței dintre momentele de timp la care se fac măsurătorile.

- **Contribuții aplicative**

- Bazată pe existența unei varietăți bogate de coduri detectoare şi corectoare de erori, se propune o soluție modernă şi elegantă de realizare a sistemelor automate tolerante la defecțiuni.
- Analiza mediului de calcul informațional autonom, ca bază a construirii sistemelor inteligente de management informatic cu aplicații industriale, pornind de la soluțiile dezvoltate in mediile academice.
- Prezentarea unei structuri hardware minimale, pentru calculatoarele numerice autonome, conținând: senzori, traductoare, controlere, actuatori, conectați la sisteme puternice de calcul, prin similitudine cu echipamentele de control inteligent automat, cu funcționare în timp real.
- Pornind de la conceptul modelării şi optimizării funcției de bază a componentelor, s-a propus o soluție novativă care permite generarea de componente independente, reutilizabile, capabile să preia funcțiile componentelor defecte.
- Elaborarea contribuțiilor specifice modelelor „platformă independentă" şi „platformă specifică", precum şi elementele necesare pentru generarea codului executabil, pornind de la diagrama stărilor care gestionează aceste modele.
- În scopul creşterii performanțelor la costuri accesibile s-a propus şi anallizat o arhitectură modernă, utilizând structurile autonome computerizate adaptive, organizate ierarhic.
- Identificarea şi stabilirea nucleului modelelor predictive de control adaptiv al sistemelor autonome de calcul.
- A fost elaborată o schemă originală de model predictiv de control alcătuită din două subsisteme în care semnalele la ieşirea blocului de prelucrare a semnalelor neliniare se aplică la intrarea unui bloc de inversare care anulează neliniaritatea.
- Elaborarea unui model de control predictiv desensibilizat ca alternativă viabilă la strategiile robuste propuse în literatura de specialitate, pentru modelele predictive de control adaptiv.
- Implementarea algoritmilor de predictie in arhitectura sistemelor autonome de calcul şi evaluarea erorilor de modelare produse la predicție.
- Pe baza suportului matematic dezvoltat, s-a imaginat un algoritm de predicție şi estimare a erorilor, aplicabil la optimizarea sistemelor digitale de control adaptiv.
- Aplicarea relațiilor deduse pentru cele trei tipuri de predicție, la studiul concret al unui sistem de control adaptiv, a demonstrat că poate fi îndeplinită conditia de estimare optimală, cu erori minime, dacă se aplică algoritmul de predicție pe interval fix.
- Elaborarea modelului estimativ pentru datele înregistrate pe durata unei săptămâni, a furnizat informațiile necesare procesului de antrenare prin învățare a sistemului de estimare prin predicție.

- Au fost trasate grafice intuitive care ilustrează dependența temporală a parametrilor modelului de estimare propus.
- Conceperea unei arhitecturi optimale (complexitate/preț de cost) de sistem autoadministrat, capabil să se adapteze testelor de mentenanță, utilizând programe dedicate disponibilă la IBM.
- Asocierea noțiunii de client cu un generator de trafic virtual capabil să modifice rata de emisie între o distribuție liniară şi una Gaussiana, pe baza unei cereri http URL pentru o adresă specificată.
- Implementarea buclei de calcul autonom TIO la controlul şi ajustarea dinamică a numărului optim de calculatoare participant la aplicație.
- Programarea buclei de control pentru a estima efectul adăugării/izolării unui server în diferite configurații de operare.
- O analiză comparativă a performanțelor care se obțin prin utilizarea unui controler cu prag(simplu şi ieftin) sau a unui controler complicat cu reacție feed-forward(complicat cu timp mare de autoreglare).
- Experimentele au arătat că este posibil să se selecteze o varietate bogată de modele de control dependente de distribuția cererilor de utilizare.
- S-a demonstrat experimental că nu este avantajos să se păstreze acelaşi tip de controler pe toată durata rulării programului de procesare, atunci când distribuția cererilor de utilizare este variabilă.
- Pentru automatizarea procesului de adăugare/înlăturare a unui server din cluster pe baza calculării obiectivelor de performanță; elementul de control TIO a fost completat cu un modul adițional denumit Analizor de Obiectiv.
- Elaborarea unei arhitecturi experimentale de tip grid capabilă să funcționeze atât cu sarcini interactive cât şi pentru încărcări de tip batch.
- Implementarea software a unui algoritm de testare pantru estimarea probabilității de apariție a erorilor de încălcare a obiectivelor în sistemele grid prevăzute cu AO.
- Măsurări experimentale privind performanțele sistemului cu/fără TIO în condiții de test diferite, interpretarea rezultatelor şi concluziile ce se impun.
- Conceperea unui ansamblu de test simplificat pentru o aplicație de tip Domino Server folosită de poşta electronică la IBM, utilizând teoria controlului automat în monitorizarea sistemelor complexe de calcul.
- Simularea controlului automat pentru un Domino Server utilizând programul “spreadsheet “care rezolvă ecuații cu diferențe finite şi afişează rezultatele funcție de timp.

- Elaborarea unui model comun de analiză pentru TIO Global Resource Manager şi WebSphere Extended Deployment care simulează funcționarea în condițiile în care spațiile de lucru se schimbă dinamic.

- S-a demonstrat experimentul că pentru spațiile de lucru variabile în timp alocarea/dealocarea rapidă de server la cluster, nu este sesizabilă, datorită controlului precis al întârzierilor de alocare.

- Elaborarea unei arhitecturi experimentale la nivel de componente pentru un sistem autonom de calcul adaptiv robust, utilizat pentru controlul activ al sarcinii unității centrale de procesare într-un mediu cu servere virtuale.

- Simularea unui server virtual cu ajutorul programului VMM instalat pe un calculator performant dotat cu procesor Intel Core 2 Duo.

- Proiectarea şi implementarea software a unei interfețe grafice cu utilizatorul care configurează sistemul de test utilizând biblioteca Java Server Faces.

- Calculatoarele virtuale, actuatorii, senzorii, filtrul, estimatorii şi blocul decizional, au fost simulate prin paravirtualizare utilizând sistemul de operare Debian Etch Linux disponibil la IBM.

6.2 CONCLUZII FINALE

Creşterea complexității proceselor industriale şi dezvoltarea explozivă a tehnicii digitale de calcul, implică alocarea de resurse suplimentare gestionate în centrele de date.Strategiile actuale se bazeaza pe comportamente formale si utilizeaza componente inteligente capabile să ia decizii legate de alocarea optimală a resurselor disponibile. Abordarea clasică pentru asignarea resurselor necesare unei anumite aplicații constă în furnizarea de date suficiente perioadei de vârf în mod anticipat – alocarea statică a resurselor(ineficientă şi costisitoare). Soluția modernă alternativă, constă în alocarea dinamică a resurselor şi izolarea lor din mediul de lucru, dacă solicitarea este redusă. Alocarea sau dealocarea resurselor se face pe baza unui algoritm de optimizare, conceput să determine extremul valorii unei funcții de utilitate. Un sistem de gestionare adaptivă a datelor va trebui să fie capabil să se adapteze în timp real la schimbarea strategiei, fără să necesite o recompilare a codului de bază.

Sistemele actuale de calcul disponibile sunt nesigure şi dificil de gestionat, datorită complexității infrastructuii informaționale care înglobează simultan aplicații hardware, software, dar şi servicii web în rețele interconectate. O soluție viabilă pentru automatizarea structurii informatice şi managementul optimal al funcționării ei, o constituie elaborarea sistemelor autonome de calcul reconfigurabil, capabile să se autogestioneze(autoconfigureze, autooptimizeze, autorepare şi autoprotejeze).

În acest domeniu actual şi generos de cercetare fundamentală şi aplicativă se încadrează preocupările şi rezultatele raportate în această teză de doctorat. Structurată pe şase capitole conținând două anexe şi o bibliografie selectivă de actualitate, teza de doctorat elaborată prezintă importanța

şi necesitatea dezvoltării acestui domeniu de cercetare, stadiul actual al realizărilor, contribuțiile teoretice şi aplicative aduse de autor la conceperea unor modele şi arhitecturi originale de echipamente autonome, reconfigurabile, tolerante le defecțiuni.

Din analiza structurilor teoretice şi aplicative dezvoltate pe parcursul elaborării tezei de doctorat, se desprind următoarele concluzii finale prezentate sintetic pe capitole:

- Capitolul 1. Implementarea structurilor redundante protective la defecțiuni, reprezintă cea mai modernă soluție de creştere a fiabilității, mentenabilității şi disponibilității echipamentelor electronice, digitale complexe. Alegerea strategiei optimale de aplicare a redundanței protective la defecțiuni, are la bază o analiză complexă privind importanța aplicației şi variației funcției de fiabilitate raportată la costul solicitat.
- Capitolul 2. Calculatoarele autonome actuale tind să înglobeze structuri automate în sistemele de management informatic capabile să se poată adapta schimbărilor de configurație, protecție şi să utilizeze optimal resursele disponibile. Prin adăugarea unui ciclu suplimentar de control automat, având ca obiectiv gestionarea resurselor sistemului, se poată asigura reconfigurarea în timp real şi adoptarea unei noi bucle optimale de control automat.
- Capitolul 3. Analiza matematică a modelelor predictive de control adaptiv implementate în sistemele autonome de calcul, a demonstrat că pentru a obține un algoritm optimal, este necesar să se soluționeza probleme complexe, care pot fi simplificate numai în condiții inițiale specificate. Abordarea unui model predictiv de control neliniar în spațiul multidimensional nu oferă satisfacții, deoarece implică un consum mare de resurse de calcul şi implicit – costuri ridicate la prelucrare.
- Capitolul 4. La intrările sistemelor digitale de calcul autoadministrate sunt prezente zgomote şi ca urmare, este necesară utilizarea modelelor de control predictiv, pentru analiza şi controlul parametrilor de proces. Calculele analitice şi simulările dezvoltate în cadrul acestui capitol au demonstrat că utilizarea algoritmilor de predicție pentru estimarea erorilor, reprezintă o solutie viabilă de creştere a performanțelor şi scădere a costurilor solicitate de sistemele autonome de calcul autoadministrate.
- Capitolul 5. Cele cinci arhitecturi de sisteme digitale autoadministrate propuse pentru testele de verificare şi diagnoză, au furnizat informații relevante care consolidează suportul matematic al dezvoltărilor teoretice din teza de doctorat. Pentru a obține facilități sporite la testare cu costuri scăzute, s-a optat pentru simulare. Pe fiecare suport hardware al calculatoarelor conectate la cluster, au fost simulate servere virtuale diferite a căror funcționare a fost gestionată de sistemul de calcul autonom, adaptiv, autoadministrat testat.
- Anexele tezei de doctorat prezintă câteva din programele sursă scrise în limbaj Java care implementează conceptele teoretice dezvoltate în teză şi care au permis studierea comportării dinamice a unui sistem de calculatoare autoadministrate dintr-un centru de calcul modern.

Teza de doctorat prezentată, sintetizează principalele preocupări de cercetare ştiintifică ale autorului în domeniul de mare acutalitate al sistemelor digitale de calcul autoadministrate. Studiile şi realizările prezentate în teză sunt reprezentative pentru activitatea de cercetare desfăşurată în ultimii ani la companiile Think Dynamics Canada şi IBM Canada şi USA. Sub coordonarea tehnică a autorului, colective de cercetare de la companiile amintite, au dezvoltat produsele software: „Tivoli Provisioning Manager" şi „Tivoli Intelligent Orchestrator" care stau la baza conceperii arhitecturii unor sisteme digitale autonome produse la IBM. Rezultatele parțiale ale cercetărilor au fost publicate de autor şi colaboratori în diverse Edituri Tehnice, proceedingurile unor manifestări ştiințifice indexate în baze de date internaționale sau în Rapoarte de cercetare ştiințifică la IBM Canada şi USA.

6.3 CERCETĂRI DE PERSPECTIVĂ

Dinamica dezvoltării tehnologice actuale, a condus la apariția de noi echipamente hardware de conducere a proceselor industriale şi dezvoltarea de noi pachete software dedicate aplicațiilor de: modelare, supraveghere, optimizare şi management a infrastructurii IT. Singura soluție viabilă de creştere a fiabilității, mentenabilității şi disponibilității sistemelor numerice de calcul ,o constituie elaborarea de platforme inteligente, capabile să se autoadministreze, autoadapteze şi autoconfigureze, utilizând structuri redundante protective la defecțiuni. Dintre direcțiile viitoare în care pot evolua cercetările de perspectivă în domeniul sistemelor electronice capabile să se autogestioneze putem enumera:

- Dezvoltarea unor metodologii de modelare (pachete software) utilizând sisteme ce supervizează evenimente aleatoare distribuite.
- Identificarea unor modele generice de referință, capabile să acopere în mod unitar cerințele actuale si de perspectiva ale serviciilor web autonome şi autoadministrate.
- Elaborarea unor sisteme evolutive, autoadaptabile capabile să se reconfigureze pe baza unor noi pachete software autoconfigurabile.
- O nouă concepție în dezvoltarea arhitecturii sistemelor autonome de calcul reconfigurabile, având un nivel ridicat de abstractizare, cu potențial scalabil şi posibilități de integrare care să satisfacă o arie lungă de aplicatii.
- Conceperea unor mecanisme specifice pentru generarea fişierelor autoadaptabile care să conțină toate informațiile solicitate de execuția tranzițiilor în conducerea ierarhizată a proceselor industriale.

Considerăm că am prezentat numai câteva idei privind posibilele direcții de cercetare viitoare, în domeniul sistemelor autonome autoadministrate. Lista prezentată poate fi completată cu generozitate de cercetătorii avizați din acest domeniu atractiv de cercetare.

ANEXE

ANEXA 1 COD SURSA AL BIBLIOTECII JAVA DE PROTOTIPURI 2SE 1.4

În Anexa 1 prezentam cateva programe sursa in Java care implementeaza conceptele descrise in teza. Programele au facut parte din produsul software Think Control dezvoltat de catre compania Think Dynamics sub conducerea autorului, care a fost convertit in produsele software Tivoli Provisioning Manager si Tivoli Intelligenet Orchestrator in cadrul IBM, de asemenea sub conducerea tehnica a autorului.

Progamele incluse sunt:

1.1. Clasa Low Pass Filter

1.2. Clasa Idle Server Cost

1.3. Clasa Breach Probability Implementation

1.4. Clasa Service Breach Cost

1.5. Clasa App.Classifier Bean

1.6. Clasa Service Level Objective Set

1.7. Clasa Workload Cluster

1.8. Clasa Queue Network Processor

1.9. Clasa App.Workload Model

1.10. Clasa ANOVA Predictor

1.11. Clasa Weekly ANOVA

1.12. Clasa ANOVA Model

1.13. Clasa Abstract Processor

1.14. Clasa App.Controller Bean

1.15. Clasa Queue Hybrid Processor

1.
```
class LowPassFilter extends Filter {
 /** filter coefficients */
private double[] inputCoEffs;

/** filter coefficients */
private double[] outputCoEffs;

/** first cell in input/output arrays */
private int prev;

/** minimum output value; -1 means no limit */
private double min;

/** maximum output value; -1 means no limit */
private double max;

/** desired output precision; -1 means no rounding */
private double threshold;

/** filter memories */
private double[] prevInput;

/** filter memories */
private double[] prevOutput;

/** Construct filter */
public LowPassFilter(MetricContext context,
                     MetricFilter parent,
                     double min, double max, double threshold,
                     double[] inputCoEffs,
                     double[] outputCoEffs) throws SubscriptionException {
    super(context, parent);
    this.min = min;
    this.max = max;
    this.threshold = threshold;
    this.inputCoEffs = inputCoEffs;
    this.outputCoEffs = outputCoEffs;
    prev = inputCoEffs.length;
    prevInput = new double[inputCoEffs.length];
    prevOutput = new double[inputCoEffs.length];
}

/** calculate filtered value */
public double value() {

    // should this shortcut be a class attribute?
    final int order = inputCoEffs.length;

    // push new values in history buffer
    prev = prev == 0? order-1: prev-1;
    prevInput[prev] = child.value();
    prevOutput[prev] = 0;

    //
    double weightedInputs = 0;
    double weightedOutputs = 0;
    for (int j = 0; j < inputCoEffs.length; j++) {
        weightedInputs += inputCoEffs[j] * prevInput[(prev + j) % order];
    }
    for (int j = 1; j < inputCoEffs.length; j++) {
        weightedOutputs += outputCoEffs[j] * prevOutput[(prev + j) % order];
    }

    // Calculate the new value and add it to the list
    prevOutput[prev] = (weightedInputs - weightedOutputs) / outputCoEffs[0];

    // adjust value to be in [min, max] interval
    double value = prevOutput[prev];
    if (min >= 0.0 && min > value) { value = min; }
    else if (max >= 0.0 && max < value) { value = max; }
```

```
        // adjust value to be abs(value) > threshold
        if (Math.abs(value) < threshold) {
            value = 0.0;
        }

        return value;
    }

    //
    // Xml representation
    //
    public String getXmlString() {
        String out = "<low-pass-filter context='"+context.toString()
                +"' min='"+min+"' max='"+max+"' threshold='"+threshold+"'>";
        out += child.getXmlString();
        out += "</low-pass-filter>";
        return out;
    }
}
```

2.

```
public class IdleServersCost implements CostFactor {
/**
 * Description:  Cost for leaving some of the servers idle (i.e. Not saturating usage)
 */
    public static Category log = Category.getInstance(IdleServersCost.class.getName());
    private Integer idleServers;
    public IdleServersCost() {
    }

    /**
     * Set the number of idles servers in the global pool for now.
     */
    public void setIdleServers(Integer idleSvr) {
        idleServers = idleSvr;
    }

    /**
     * @param edgeChange The change on the configuration
     * @param costEnv    Other environment factors which may change the cost
     *
     * @return The cost of moving from one node to the other
     */
    public double getCost(InfrastructureChange edgeChange, CostEnvironment costEnv) {
/*
        double exponent, adjustment;

        exponent = 1.8;
        adjustment = 0;

        double dblSvrCost = Math.pow((double) idleServers.intValue(), exponent) + adjustment;

        if (edgeChange.getDeltaServer() > 0)
            return -dblSvrCost;
        else
            return dblSvrCost;
        double mathlogbase, scale, adjustment;
        mathlogbase = 1.8;
        scale = 50;
        adjustment = -50;
        double origNumber = idleServers.intValue();
        double dblIdleCost = 0;

        if (edgeChange.getDeltaServer() > 0) { // Adding servers, idleServers is the resultant
            dblIdleCost = Math.log((double) origNumber + costEnv.getIdleServerDifference())
                / Math.log(mathlogbase) * scale - adjustment;
        }
        else { // Removing servers, idleServers is also the resultant
            dblIdleCost = - (Math.log((double) origNumber + costEnv.getIdleServerDifference() +1)
                / Math.log(mathlogbase) * scale + adjustment);
```

```
        }
*/

        double dblIdleCost = 0;

        // Idle server cost formulat as follows:
        // Cost of idle = (scale / (((# of Idle servers / slope) + 1) ^ exponent)) - adjustment
        double scale, slope, exponent, adjustment;
        exponent    = 6.300286641;
        scale       = 290;
        adjustment  = 0;
        slope       = 98;

        // Get net changes


        int iIdle = idleServers.intValue();
        if (edgeChange.getTotalServersChanged() < 0) { // Removing a server should have reverse
effect
            iIdle--; // Use the reduced servers values
        }

        int iDiff = costEnv.getIdleServerDifference();
        int projectedNumberOfIdleServers = iIdle + iDiff;
        dblIdleCost = scale / Math.pow((projectedNumberOfIdleServers / slope) + 1,
            exponent) - adjustment;
        if (edgeChange.getTotalServersChanged() < 0) { // Removing a server should have reverse
effect
            dblIdleCost *= -1;
        }

        log.debug("idleServers="+idleServers+
            " iDiff="+iDiff+
            " IdleServerCost=" + dblIdleCost);

        return dblIdleCost;
    }
}
```

3.

```
class BreachProbabilityFunctionImpl implements BreachProbabilityFunction {
/**
 * Description:  Main implementation for the BreachProbabilityFunction interface<br>
 */
    public static Category log =
Category.getInstance(BreachProbabilityFunctionImpl.class.getName());

    private double errorVariance;
    private ServiceLevelObjectiveSet serviceLevelObjectives;
    private InfrastructureNeeds parent;
    private int clusterId;
    private int webClusterId;

    public BreachProbabilityFunctionImpl(
            int clusterId,
            int webClusterId,
            InfrastructureNeeds parent,
            double errorVariance,
            ServiceLevelObjectiveSet serviceLevelObjectives) {
        this.clusterId = clusterId;
        this.webClusterId = webClusterId;
        this.parent = parent;
        this.errorVariance = errorVariance;
        this.serviceLevelObjectives = serviceLevelObjectives;
    } //-- Ctor

    public double classifyBreachProbability(int numberOfServers) {
        TimeStampedValueSet currentValueSet = parent.getCurrentValueSet();
```

```
        double criticalCPUUtilization = 1.0;
        try {
            criticalCPUUtilization = serviceLevelObjectives.get(
                clusterId, SignalType.CRITICAL_CPU_UTILIZATION);
        } catch (Exception e) {
        }

        double SLABreachProbability = 0;
        double currentArrivalRate = currentValueSet.getClusterValue(webClusterId,
            SignalType.ARRIVAL_RATE.getName()).getValue();
        if (currentArrivalRate != 0) {
            double currentCPUUtilization = currentValueSet.getClusterValue(clusterId,
                SignalType.CPU_UTILIZATION.getName()).getValue();
            double currentNumberOfServers = currentValueSet.getClusterValue(clusterId,
                SignalType.NUMBER_OF_SERVERS.getName()).getValue();

            TimeStampedValueSet predictedValueSet = parent.getPredictedValueSet();
            double arrivalRate = predictedValueSet.getClusterValue(webClusterId,
                SignalType.ARRIVAL_RATE.getName()).getValue();

            double upperArrivalRate = convertCPUUtilizationToArrivalRate(
                    currentArrivalRate,
                    currentCPUUtilization,
                    currentNumberOfServers,
                    criticalCPUUtilization,
                    numberOfServers);

            SLABreachProbability =
                cumulativeNormalFunction((arrivalRate - upperArrivalRate) /
                                         errorVariance);

            if (SLABreachProbability > 1) {
                SLABreachProbability = 1;
            }
        }


        log.debug ("clusterId="+clusterId+
            ", errorVariance="+errorVariance+
            ", currServers="+currentValueSet.getClusterValue(clusterId,
                SignalType.NUMBER_OF_SERVERS.getName()).getValue()+
            ", numServers="+numberOfServers+
            ", BP="+SLABreachProbability);

        return SLABreachProbability;
    } //-- classifyBreachProbability


    /**
     * Converts CPU Utilization to Arrival Rate, taking into account different number of servers
for
     * the current CPU Utilization and the CPU Utilization that has to be converted.
     */
    private double convertCPUUtilizationToArrivalRate(
                double currentArrivalRate,
                double currentCPUUtilization,
                double currentNumberOfServers,
                double convertCPUUtilization,
                double convertNumberOfServers) {
        if (currentCPUUtilization == 0) {
            currentCPUUtilization = 0.01;
        }
        return (currentArrivalRate * convertNumberOfServers * convertCPUUtilization) /
                        (currentNumberOfServers * currentCPUUtilization);
    } //-- convertCPUUtilizationToArrivalRate


    /**
     * Calculates the cumulative normal function in x.
     * The following numerical method was taken from http://www.pitt.edu/~wpilib/cmt.html
```

```
The standard (mean=0 and variance=1) normal density function is
<br><br>
Z(x) = 1/sqrt(2*pi)*e^(-(1/2)*x^2)<br>
<br><br><br><br>
While it is not possible to write a solution to this in a simple,
closed form expression not involving infinite sums, there are many
good approximations available. It is not necessary to resort to
numerical integration techniques such as trapezoidal or Simpson's
rule. In fact, far more accuracy can be achieved through one of the
approximations which can be found in the references listed below.
<br><br>
One simple approximation to P(X), the cumulative normal, is:
(from Abramowitz and Stegun)
<br><br>
Let t = 1/(1+pX)
<br><br>
where p=0.33267
<br><br>
then
<br><br>
P(x) = 1-Z(x)*[a_1*t + a_2*t^2 + a_3*t^3]
<br><br>
where
<br><br>
a_1 =  0.4361836<br>
a_2 = -0.1201676<br>
a_3 =  0.9372980<br>
<br><br>
The error in this approximation is less than 0.00001 for all X.
<br><br>
Other approximations can be found in Abramowitz & Stegun; Press, et al;
and in Hart, et al. In particular, Hart has a very extensive set of
approximations, some with accuracy over 20 significant digits within
a given region.
     */
    private double cumulativeNormalFunction(double x) {
        if (x < 0) {
            return 1 - cumulativeNormalFunction(-x);
        }

        double a1 =  0.4361836;
        double a2 = -0.1201676;
        double a3 =  0.9372980;

        double z = Math.exp(- x * x / 2) / Math.sqrt(2 * Math.PI);
        double t = 1 / (1 + 0.33267 * x);

        return 1.0 - z * t * (a1 + t * (a2 + t * a3)) ;
    } //-- cumulativeNormalFunction
}
```

4.

```
public class ServiceBreachCost implements CostFactor {
/**
 * Description:  Cost of breaching the SLA or PLA<br>
 */
   public static Category log = Category.getInstance(ServiceBreachCost.class.getName());

    private HashMap hmAppPriority;  // Priority of the cluster against cluster id
    private HashMap hmAppPerformance; // Performance for the application
    private HashMap hmAppServers; // Number of servers the application uses

    /**
     * Setup the priority of the hash map for the clusters.
     * The priority could be a non-integer but cannot be negative.
     *
     * @param newPriority The hashmap priority (type double) for individual clusters.
     * The smaller number the higher priority.  Range of priority falls between 1 and 10.
     */
```

```
    public void setAppPriority(HashMap newPriority) {
        hmAppPriority = newPriority;
    }

    /**
     * Setup the performance of the hash map for the clusters.
     *
     * @param newPerformance The hashmap priority (type double) for individual clusters
     * between -100% to 100%.
     */
    public void setAppPerformance(HashMap newPerformance) {
        hmAppPerformance = newPerformance;
    }

    /**
     * Setup the number of servers in the hash map for the clusters.
     *
     * @param numServers Number of servers
     */
    public void setServerNumbers(HashMap newNumServers) {
        hmAppServers = newNumServers;
    }

    /**
     * @return Priority factor for the service.
     */
    private double getPriorityFactor(InfrastructureChangeItem edgeChange) {
        // Formula:
        //      Priority factor = 11 - priority
        Double dblPriority = (Double) hmAppPriority.get(
            new Integer(edgeChange.getClusterid()));
        if (dblPriority == null) {
            throw new KanahaSystemException(ErrorCode.nullPointer,
                    "Missing application priority for ClusterID=" + edgeChange.getClusterid());
        }
        double outPriorityCost = 11 - dblPriority.doubleValue();


        log.debug("priority="+dblPriority+" getPriorityFactor returns: " +
String.valueOf(outPriorityCost));
        return outPriorityCost;
    }

    /**
     * @return Probability of breach factor for the service.
     */
    private double getBreachFactor(InfrastructureChangeItem edgeChange, int iExpChange) {
        // Formula:
        //      Performance cost = Priority * 1000
        double baseCost, exponent, adjust;

        BreachProbabilityFunction bpf =
            (BreachProbabilityFunction) hmAppPerformance.get(new
Integer(edgeChange.getClusterid()));

        // Get the number of Servers (prior to current change) for which to determine the cost
        Double dblServers = (Double) hmAppServers.get(
            new Integer(edgeChange.getClusterid()));
        double currentSvrs = dblServers.doubleValue() + iExpChange - edgeChange.getDeltaServer();

        double dblPerform = bpf.classifyBreachProbability((int)currentSvrs);
        double outPerfCost = dblPerform * 1000;

        log.debug("BreachFactor=" + String.valueOf(outPerfCost));
        return outPerfCost;
    }

    /**
     * Encourage assignment of servers by the rate of return.
     *
```

```
     * @param edgeChange The change for the cost to calculate on
     * @param iExpChange Excpected changes (total) so far in traversal for the specified cluster
     *
     * @return Cost compared to others for efficiency gained
     */
    private double getRateOfReturnFactor(InfrastructureChangeItem edgeChange, int iExpChange) {
        double currentSvrs = 0;

        Double dblServers = (Double) hmAppServers.get(
            new Integer(edgeChange.getClusterid()));
        // New number of servers...
        currentSvrs = dblServers.doubleValue() + iExpChange;
        if (edgeChange.getDeltaServer() > 0) {
            currentSvrs--; // Calculate from original number of servers
        }

/*
        // Adding from x to x+1 and removing form x+1 to x should have the same cost
        if (edgeChange.getDeltaServer() < 0) {
            if (currentSvrs != 0)
                currentSvrs--; // Conservation of cost adding and removing servers
        }
*/

        double rorFactor = 2 - currentSvrs /(currentSvrs + 1);
        if (edgeChange.getDeltaServer() < 0) { // Remove server scale has to be calculated
accordingly
            rorFactor *= (currentSvrs+1) / currentSvrs;
        }

        log.debug("clusterServers="+dblServers.doubleValue()+" netServerChange="+iExpChange+
            " RateOfReturnFactor=" + rorFactor);

        return rorFactor;
    }

    /**
     * @param edgeChange The change on the configuration
     * @param costEnv    Other environment factors which may change the cost
     *
     * @return The cost of moving from one node to the other
     */
    public double getCost(InfrastructureChange edgeChange, CostEnvironment costEnv) {
        double weightApp = 0.05;

        Collection changeCol = edgeChange.getInfrastructureChangeItem();
        Iterator iter = changeCol.iterator();
        double totalCost = 0;

        while (iter.hasNext()) {
            InfrastructureChangeItem currentChange = (InfrastructureChangeItem) iter.next();

            double dPriorFactor = getPriorityFactor(currentChange);
            double dBreachFactor = getBreachFactor(currentChange,
costEnv.getClusterServersDiff());
            double dRorFactor = getRateOfReturnFactor(currentChange,
                costEnv.getClusterServersDiff());

            double tempCost = weightApp * dPriorFactor * dBreachFactor * dRorFactor; // Negative
cost for calculations
            if (currentChange.getDeltaServer() > 0) { // Removing a server should have reverse
effect
                tempCost *= -1;
            }
            totalCost += tempCost;
        }

        log.debug("Total cost for Breach=" + totalCost);
        return totalCost;
    }
```

```
}
```

5.
```
class AppClassifierBean implements AppClassifier, AppClassifierBeanMBean {

private static Category log = Category.getInstance(AppClassifierBean.class.getName());

    private Application application = null;
    private Collection clusters = null;
    private Predictor predictor = null;
    private int webClusterId;

    private boolean learningEnabled = true;

    public void init(Application application, Collection clusters, Predictor predictor) {
        this.application = application;
        this.clusters = clusters;
        this.predictor = predictor;

        /**
        // As a workaround will assume
        // that first cluster returned from dcm is a web-tier cluster,
        // and second cluster is an app-tier cluster.
        * */
        Iterator tiers = clusters.iterator();
        webClusterId = ((Cluster) tiers.next()).getId();
    }

    public InfrastructureNeeds classifyInfrastructureNeeds(
                TimeStampedValueSet currentValues,
                TimeStampedValueSet predictedValues,
                ServiceLevelObjectiveSet serviceLevelObjectives)
            throws ClassifierException {

        Iterator iterCluster = clusters.iterator();
        while (iterCluster.hasNext()) {
            Cluster cluster = (Cluster) iterCluster.next();

            double currentNumberOfServers =
                currentValues.getClusterValue(
                    cluster.getId(), SignalType.NUMBER_OF_SERVERS.getName()).getValue();

            double numberOfNeededServers = currentNumberOfServers;

            if (cluster.isManaged()) {
                double criticalCPUUtilization = 0;
                try {
                    criticalCPUUtilization = serviceLevelObjectives.get(
                        cluster.getId(), SignalType.CRITICAL_CPU_UTILIZATION);
                } catch (Exception e) {
                    throw new ClassifierException(ErrorCode.noSLACPUUtilization);
                }

                double currentCPUUtilization = currentValues.getClusterValue(cluster.getId(),
                    SignalType.CPU_UTILIZATION.getName()).getValue();
                double currentArrivalRate = currentValues.getClusterValue(webClusterId,
                    SignalType.ARRIVAL_RATE.getName()).getValue();
                double predictedArrivalRate = predictedValues.getClusterValue(webClusterId,
                    SignalType.ARRIVAL_RATE.getName()).getValue();

                if (currentArrivalRate < 0.01) {
                    // this is to prevent noise causing wild swings in neededServers
                    // when Arrival Rate and predicted ArrivalRate are close to zero.
                    numberOfNeededServers = currentNumberOfServers *
                        (currentCPUUtilization / criticalCPUUtilization);
                }
                else {
                    numberOfNeededServers = currentNumberOfServers *
                        (currentCPUUtilization / criticalCPUUtilization) *
```

```
                        (predictedArrivalRate / currentArrivalRate);
                }

                log.debug ("clusterId="+cluster.getId()+
                    " currentCPU="+currentCPUUtilization+
                    " critCPU="+predictedValues.getClusterValue(cluster.getId(),
                        SignalType.CRITICAL_CPU_UTILIZATION.getName()).getValue()+
                    " neededServers="+numberOfNeededServers);
            }
            // this is needed by the InfrastructureNeeds.getClusterNeeds() method
            // which is used by the ResourceBroker to fetch numberOfNeededServers.
            predictedValues.setClusterValue(cluster.getId(),
                SignalType.NUMBER_OF_SERVERS.getName(), numberOfNeededServers);
        }

        InfrastructureNeeds infrastructureNeeds = InfrastructureNeedsFactory.getInstance(
                application, clusters, currentValues, predictedValues);

        double errorVariance = 0;
        if (predictor.isReady()) {
            try {
                errorVariance = predictor.getErrorVariance(
                            infrastructureNeeds.getCurrentTime(),
                            infrastructureNeeds.getPredictionTime());
            } catch (PredictorException ex) {
                throw new ClassifierException(ErrorCode.cannotCalculateBreachProbability, ex,
application.getName());
            }
        } else {
            /**
             * if the predictor is not ready, use the variance of
             * the raw arrival rate signal.
             */
            errorVariance = 1.0;
            TimeStampedValue arrivalRateVariance = predictedValues.
                getClusterValue(webClusterId,
                SignalType.ARRIVAL_RATE_VARIANCE.getName());

            if (arrivalRateVariance.isAvailable()) {
                errorVariance = arrivalRateVariance.getValue();
                if (errorVariance < 0.1)
                    errorVariance = 0.1;
            }
        }
        // Create the BreachProbabilityFunction for each tier for this prediction interval
        iterCluster = clusters.iterator();
        for (int iTier = 0; iterCluster.hasNext(); iTier++) {
            Cluster cluster = (Cluster) iterCluster.next();
            int clusterId = cluster.getId();

            if (cluster.isManaged()) { // ((iTier == Tier.APPLICATION) || (iTier == Tier.WEB)) {
                // Create the function for the app-tier only
                BreachProbabilityFunction bpf = new BreachProbabilityFunctionImpl(
                                    clusterId, webClusterId,
                                    infrastructureNeeds,
                                    errorVariance,
                                    serviceLevelObjectives);
                infrastructureNeeds.setBreachProbabilityFunction(clusterId, bpf);
            } else {
                /** Create function for the other tiers */
                infrastructureNeeds.setBreachProbabilityFunction(clusterId, null);
            }
        }
        return infrastructureNeeds;
    } //-- classifyInfrastructureNeeds

    // MBean related methods go below

    public boolean isLearningEnabled() {
        return learningEnabled;
```

```
    }
}
```

6.

```
public class ServiceLevelObjectiveSet {

/**
 * Description:  Holds the internal Service Level Objectives for one Application.<br>
 */
    private static final String ID_SEPARATOR = "-";
    private Date timeStamp = null;
    private ApplicationUC dcm;
    /**
     * Holds the SLO values (Double) indexed by a String composed of ClusterID
     * and ObjectiveType.
     */
    private HashMap hmSloValues = new HashMap();
    /**
     * This collection is initialized with all the ID's of managed
     * Clusters for this Application. This is used for retreiving
     * value from the WorkLoadModel's SignalValueSet.
     */
    private ArrayList clusterIdList = new ArrayList();
    /**
     *  List of Objective Types that we wish to keep track of.
     */
     public final static SignalType[] objectiveTypes = {
        SignalType.CRITICAL_CPU_UTILIZATION,
        SignalType.CRITICAL_ARRIVAL_RATE,
        SignalType.CRITICAL_RESPONSE_TIME };
    /**
     * Initializes the object for default values for all the
     * Clusters in on Application.
     *
     * @param dcm access object for Data Center Model
     * @param clusters collection of clusters for this application
     */
    public ServiceLevelObjectiveSet(ApplicationUC dcm, Collection clusters) {
        this.dcm = dcm;
        Iterator iter = clusters.iterator();
        while (iter.hasNext()) {
            Cluster cluster = (Cluster) iter.next();
            int clusterId = cluster.getId();
            // setup some default values for now.
            String key = composeHashKey(clusterId, SignalType.CRITICAL_CPU_UTILIZATION);
            hmSloValues.put(key, new Double (0.6));
            clusterIdList.add(new Integer(clusterId));
        }
    }

    /**
     * Load this ServiceLevelObjectSet from the SLO data in the database.
     * Any entries that are not in the database remain at their default
     * values set in the Constructor.
     */
    public void loadFromDCM () {
        if (dcm == null)
            return;
        Iterator iter = clusterIdList.iterator();
        while (iter.hasNext()) {
            int clusterId = ((Integer)iter.next()).intValue();
            for (int i=0; i<objectiveTypes.length; i++) {
                SignalType objectiveType = objectiveTypes[i];
                ServiceLevelObjective slo =
                    dcm.findServiceLevelObjective (clusterId, objectiveType.getName());
                if (slo != null)
                    hmSloValues.put(composeHashKey(clusterId, objectiveType),
                        new Double (slo.getValue()));
            }
        }
```

```
    }

    /**
     * Update the ServiceLevelObjectives from the WorkLoadModel results,
     * and persist to the database.
     * @param wmlResults result set produced by WorkLoadModel.
     */
    public void update (SignalValueSet wmlResults)
        throws AppControllerException
    {
        this.timeStamp = wmlResults.getTimeStamp();
//        hmSloValues.clear();
        Iterator iter = clusterIdList.iterator();
        while (iter.hasNext()) {
            int clusterId = ((Integer)iter.next()).intValue();
            for (int i=0; i<objectiveTypes.length; i++) {
                SignalType objectiveType = objectiveTypes[i];
                TimeStampedValue tsValue =
                    wmlResults.getClusterValue(clusterId, objectiveType.getName());
                if (tsValue.isAvailable()) {
                    put (clusterId, objectiveType, tsValue.getValue());
                }
            }
        }
    }

    /**
     * Set the value of one ServiceLevelObjective for a specific Cluster.
     * After storing the value in a local HashMap, we fetch the
     * corresponding SLO from the DCM, and if the new value is
     * different we persist this to the DCM
     * @param clusterId SLO is associated with this Cluster
     * @param objectiveType the type of SLO.
     * @param newValue value for SLO
     */
    public void put (int clusterId, SignalType objectiveType, double newValue)
        throws AppControllerException
    {
        String hashKey = composeHashKey(clusterId, objectiveType);
        hmSloValues.put(hashKey, new Double(newValue));
        if (dcm != null) {
            String objectiveTypeName = objectiveType.getName();
            /**
             * If the SLO already exists, we must use its id and slaId values
             * when persisting.
             */
            ServiceLevelObjective slo =
                dcm.findServiceLevelObjective(clusterId, objectiveTypeName);
            /**
             * If the SLO does not exist in the DCM, we can create a new entry
             * by setting id and slaId to -1. The DCM will generate appropriate
             * values for these.
             */
            if (slo == null) {
                ServiceLevelObjectiveType type =
                        dcm.findServiceLevelObjectiveType(objectiveTypeName, true);  // internal
                slo = new ServiceLevelObjective (-1, type.getId(), true,
                    objectiveTypeName, newValue-1, -1, new Integer(clusterId));
            }
            // persist to database.
            if (newValue != slo.getValue()) {
                slo.setValue(newValue);
                try {
                    dcm.storeServiceLevelObjective(slo);
                }
                catch (DataCenterException ex) {
                    throw new AppControllerException (ex.getErrorCode(), ex);
                }
            }
        }
```

```
    }

    /**
     * Fetch the value of one ServiceLevelObjective.
     * @param clusterId SLO is associated with this Cluster
     * @param objectiveType the type of SLO.
     */
    public double get (int clusterId, SignalType objectiveType)
        throws NoSuchElementException
    {
        Double value = (Double) hmSloValues.get(composeHashKey(clusterId, objectiveType));
        if (value == null) {
            throw new NoSuchElementException();
        }

        return value.doubleValue();
    }

    public Date getTimeStamp() {
        return timeStamp;
    }

    /**
     * creates a hashkey from clusterId and objectiveType string to be
     * used as an index into the hmValue HashMap.
     */
    private String composeHashKey (int clusterId, SignalType objectiveType) {
        return clusterId + ID_SEPARATOR + objectiveType.getName();
    }

    /**
     * Return the contents of ServiceLevelObjectiveSet as a String
     * for debugging purposes.
     */
    public String toString () {
        StringBuffer buf = new StringBuffer();
        Iterator iter = clusterIdList.iterator();
        while (iter.hasNext()) {
            double value;
            int clusterId = ((Integer)iter.next()).intValue();

            buf.append("SLO clusterId="+clusterId);
            try {
                buf.append (", critAR="+
                    get (clusterId, SignalType.CRITICAL_ARRIVAL_RATE));
            } catch (Exception e) { }

            try {
                buf.append (", critRT="+
                    get (clusterId, SignalType.CRITICAL_RESPONSE_TIME));
            } catch (Exception e) { }

            try {
                buf.append (", critCPU="+
                    get (clusterId, SignalType.CRITICAL_CPU_UTILIZATION));
            } catch (Exception e) { }

            buf.append("\r\n");
        } // end while

        return buf.toString();
    }


}
```

7.
```
public class WorkLoadCluster {
/**
 * Description:  Cluster information with WorkLoad parameters<br>
```

```
*/
    private static Category log = Category.getInstance(WorkLoadCluster.class.getName());
    private int clusterId;
    private int applicationId;
    private HashMap workLoadProperties; // Integer class ID mapped to properties
    private int totalServiceClass;
    /**
     * Initialize with original cluster object and tier work load element
     */
    public WorkLoadCluster(Cluster newAggregateCluster) {
        clusterId = newAggregateCluster.getId();
        applicationId = newAggregateCluster.getApplicationId();
    }
    /**
     * Store the property for a particular class
     *
     * @param iClassId ClassID of properties
     * @param propName Property name for values to be stored
     * @param propValue Property value to be stored
     */
    private void storeHashProperties(int iClassId, String propName,
        String propValue) {
        Properties propset = (Properties)
            workLoadProperties.get(new Integer(iClassId));
        if propset == null) {
            propset = new Properties();
            workLoadProperties.put(new Integer(iClassId), propset);
        }
        propset.put(propName, propValue);
    }
    /**
     * This method fetches the Application-level max-response-time SLO from the
     * DCM.
     * @param dcm reference to DataCenterModel
     * @param defaultValue value to use if SLO not found in DCM.
     */
    private double fetchResponseTime (ApplicationUC dcm, double defaultValue)
    {
        ServiceLevelObjectiveType sloResponseTime =
            dcm.findServiceLevelObjectiveType("max-response-time", false);
        ServiceLevelObjective slo =
            dcm.findServiceLevelObjectiveForApplication(applicationId,
                sloResponseTime.getId());
        if (slo == null) {
            log.error("max-response-time for application " + applicationId +
                " does not exist. Using default value of "+defaultValue);
            return defaultValue;
        }
        double dblMaxResponseTime = slo.getValue();
        if (dblMaxResponseTime < 0) {
            log.error("max-response-time for application " + applicationId + "=" +
                dblMaxResponseTime + " is not valid.");
            dblMaxResponseTime = 0;
        }
        return dblMaxResponseTime;
    }

    /**
     * This method fetches the Application-level min-availability SLO from the
     * DCM.
     * @param dcm reference to DataCenterModel
     * @param defaultValue value to use if SLO not found in DCM.
     */
    private double fetchAvailability (ApplicationUC dcm, double defaultValue)
    {
        ServiceLevelObjectiveType sloAvailability =
            dcm.findServiceLevelObjectiveType("min-availability", false);
        // Lookup external SLOs information
```

```
        ServiceLevelObjective slo =
            dcm.findServiceLevelObjectiveForApplication(applicationId,
                sloAvailability.getId());
        if (slo == null) {
            log.error("min-availability for application " + applicationId +
                " does not exist. Using default value of "+defaultValue);
            return defaultValue;
        }
        double dblAvailability = slo.getValue();
        if (dblAvailability < 0) {
            log.error("min-availability for application " + applicationId + "=" +
                dblAvailability + " is not valid.");
            dblAvailability = 0;
        }
        else if (dblAvailability > 1) {
            log.error("min-availability for application " + applicationId + "=" +
                dblAvailability + " is not valid.");
            dblAvailability = 1;
        }
        return dblAvailability;
    }
    /**
     * This function translates the input into properties in the structure of
     * "processer<pid>_class<cid>_<property_name>".
     * (i.e. processor1_class0_linear_scale)
     * @param dcm reference to DataCenterModel
     */
    public void storeInputParameter(ApplicationUC dcm, final int[] processorIds)
            throws WorkloadModelException
    {
        // Processing to generate WLM parameters
        // 100% Availability -> 80% CPU, 0% Availability -> 100% CPU
        double slaMaxResponse = fetchResponseTime(dcm, 0.167);
        double slaMaxCPU = 1 - 0.2 * fetchAvailability(dcm, 0.95);
        if (slaMaxCPU >= 1) {
            slaMaxCPU = .99; // Clip it such that never gets to 100% CPU
        }
        totalServiceClass = 1; /** @todo */
        workLoadProperties = new HashMap();
        Collection processorInfos = dcm.findProcessorInfosByClusterId(clusterId);
        if (processorInfos.size() == 0) {
            processorInfos = createDefaultProcessorInfos (dcm, processorIds);
        }
        for (Iterator pIter = processorInfos.iterator(); pIter.hasNext(); ) {
            ProcessorInfo info = (ProcessorInfo) pIter.next();
            String processorName = Integer.toString(info.getProcessorId());
            int iClassId = info.getWorkloadClassId();
            storeHashProperties(iClassId, "processor_info_id", String.valueOf(info.getId()));
            storeHashProperties(iClassId, "processor_id", String.valueOf(
                info.getProcessorId()));
            for (Iterator mIter = info.getProperties().keySet().iterator();
                    mIter.hasNext(); ) {
                String propName = (String) mIter.next();
                String propValue = (String) info.getProperties().get(propName);
                storeHashProperties(iClassId, propName, propValue);
            }
            // Now store the proper values
            storeHashProperties(iClassId, "slaMaxResponse",
                String.valueOf(slaMaxResponse));
            storeHashProperties(iClassId, "slaMaxCPU",
                String.valueOf(slaMaxCPU));
        }
    }

    /**
     * This method generates a collection of default ProcessorInfo objects,
     * one for each processor id.
     */
    private Collection createDefaultProcessorInfos (ApplicationUC dcm,
        final int[] processorIds) throws WorkloadModelException
```

```
    {
        ArrayList processorInfoList = new ArrayList();
        try {
            for (int i=0; i<processorIds.length; i++) {
                ProcessorInfo tempInfo = new ProcessorInfo (-1, processorIds[i], clusterId);
                int newId = dcm.createProcessorInfo(tempInfo);
                // no set for ID member, so we must create new ProcessorInfo.
                ProcessorInfo info = new ProcessorInfo (newId, processorIds[i], clusterId);
                info.setProperties(new HashMap());
                processorInfoList.add (info);
            }
        }
        catch (Exception e) {
            throw new WorkloadModelException (ErrorCode.wlmInvalidConfiguration, e);
        }
        return processorInfoList;
    }
    /**
     * Generate a static reference to the property
     *
     * @param processorId String id of processor
     * @param class Service class of interest
     * @param paramName Name of parameter
     */
/*
     public static String getFullPropertyName(String processorId,
            String serviceClass, String paramName) {
        String fullName = "processor" + processorId + "_class" +
            serviceClass + "_" + paramName;
        return fullName;
     }
*/

    /**
     * @return Properties of tiers to use
     */
    public HashMap getWorkLoadProperties() {
        return workLoadProperties;
    }

    /**
     * @return Total number of service class
     */
    public int getTotalServiceClass() {
        return totalServiceClass;
    }

    /**
     * @return cluster id
     */
    public int getId() {
        return clusterId;
    }
}
```

8.

```
public class QueueNetworkProcessor extends AbstractProcessor {

    private int numQueues;
    // The first dimension represents number of queues
    private double[][] sdArray; // Original system demand array
    private double[][] sdWorkArray; // The working array
    private int[] processerArray; // Setup number of servers per queue
    private int maxHistory = 50;
    public static int UTILIZATION_RESULT = 0;
    public static int RESPONSE_TIME_RESULT = 1;
    public static int NEXT_RESULT_INDEX = 2;
    public QueueNetworkProcessor() {
    }
```

```
    public void configureProcessor(int iProcessorId, int totalServiceClass,
            HashMap configProps[]) {
        // Generic processing
        super.configureProcessor(iProcessorId, totalServiceClass, configProps,
            maxHistory);
        // Initialize all the needed parameters
        numQueues = configProps.length;
        sdArray = new double[numQueues][totalServiceClass];
        processerArray = new int[numQueues];
        for (int iQueue = 0; iQueue < configProps.length; iQueue++) {
            for (int iClass=0; iClass < totalServiceClass; iClass++) {
                /** @todo Class ID not neccessarily 0 */
                Properties tierProperties = (Properties)
                    configProps[iQueue].get(new Integer(iClass));
                String propValue = tierProperties.getProperty("qnet_sys_demand");
                sdArray[iQueue][iClass] = java.lang.Double.parseDouble(propValue);
            }
            processerArray[iQueue] = 1; // Number of processor is default to be 1
        }
        // Get the service demand in a working array (processor # x class #)
        derivedWorkingSDArray(processerArray);
    }
    public void subscribeOutput(int outputIndex, SignalType trainingSignal)
            throws WorkloadModelException {
        super.subscribeOutputCheckIndex(outputIndex, trainingSignal,
            NEXT_RESULT_INDEX);
    }
    public double[][][] deriveOutput(double[] arrivalRates)
            throws WorkloadModelException {
        // Generic checking
        double[][][] output = super.deriveOutput(arrivalRates);
        if (output == null) {
            return null;
        }
        // Pre-determined subscription index...
        int iUtilIdx = -1;
        int iRespIdx = -1;
        int iTotalOut = 0;
        Iterator subscribeIter = subscribedList.iterator();
        iUtilIdx = findSignalIndex(UTILIZATION_RESULT);
        iRespIdx = findSignalIndex(RESPONSE_TIME_RESULT);

        // Processor specific signal generation
        // Queued network calculations
        // ===========================
        //
        double utilqueue[] = new double[sdWorkArray.length];
        for (int iQueue=0; iQueue < sdWorkArray.length;  iQueue++)
        {
            // Utilization =
            //      Summation of (Each arrival rate * system demand) for each queue
            utilqueue[iQueue] = 0;
            for (int iClass=0; iClass < totalWorkLoadClasses; iClass++)
            {
                double utilization = arrivalRates[iClass] *
                    sdWorkArray[iQueue][iClass];
                if (iUtilIdx != -1) {
                    output[iQueue][iClass][iUtilIdx] = utilization;
                }
                utilqueue[iQueue] += utilization;
            }
        }
        double[] respTime = new double[totalWorkLoadClasses];
        for (int iClass=0; iClass < totalWorkLoadClasses; iClass++)
        {
            respTime[iClass] = 0;
            for (int iQueue=0; iQueue < sdWorkArray.length;  iQueue++)
            {
                double responseTime = sdWorkArray[iQueue][iClass] /
                    (1 - utilqueue[iQueue]);
```

```
                if (iRespIdx != -1) {
                    output[iQueue][iClass][iRespIdx] = responseTime;
                }
                // Load-independent Residence time=
                //      Portion of time (out of 1) to perform single service / (1 - Utilization)
                respTime[iClass] += responseTime;
            }
        }
        return output;
    }
    public void updateModelServers(int[] servers)
            throws WorkloadModelException {
        super.updateModelServers(servers);

        // Working array is generated through here...
        processerArray = servers;
        derivedWorkingSDArray(processerArray);
    }
    public void updateModel() throws WorkloadModelException {
        // Calculate the service demand for each time in the history
        int iCPUIndex = findSignalIndex(UTILIZATION_RESULT);
        for (int iClass=0; iClass < totalWorkLoadClasses; iClass++) {
            for (int iTier = 0; iTier < totalTiers; iTier++) {
                double[] sumDemand = new double[histories[iTier].size()];
                double systemDemand = 0;
                for (int iHistory = 0; iHistory < histories[iTier].size(); iHistory++) {
                    // Calculate the service demand for each time in the history
                    sumDemand[iHistory] =
                        ((HistoryRecord) histories[iTier].get(iHistory)).
                        getHistoryData(iClass, iCPUIndex) /
                        histories[iTier].size() /
                        ((HistoryRecord) histories[iTier].get(iHistory)).
                        getArrivalRate(iClass);

                    // Add up all portions of system demand to get the total system demand
                    systemDemand += sumDemand[iHistory];

                    // Change the system demand for all classes in this tier
                    sdArray[iTier][iClass] = systemDemand;
                }
            }
        }
        derivedWorkingSDArray(processerArray);
    }

    public String getStrId() {
        /**@todo: Implement this
com.thinkdynamics.kanaha.controller.appcontroller.workloadmodel.processor.WorkLoadModelProcessor
method*/
        throw new java.lang.UnsupportedOperationException("Method getStrId() not yet
implemented.");
    }

    /**
     * Calculate the service demand array for each queue with number of processers given.<br>
     * The algorithms being used is as follows:<br>
     * <OL Type="1">
     * <LI>MP resource calculation for original system demand =<br>
     * Original System Demand (SD) /  number of resource providers
     * <LI>MP resource for delay system demand =<br>
     * Original SD * (number of resource providers - 1) / (number of resource providers)
     * </OL>
     *
     * @param processers An array of number of processors for all queues
     */
    private void derivedWorkingSDArray(int[] processers)
    {
        if (processers.length != sdArray.length)
            throw new IllegalArgumentException("Number of queues is illegal in
derivedWorkingSDArray.");
```

```
        // Caculate the system demand array for the number of processers
        int iAllQueue = 0;
        for (int iCnt=0; iCnt < processers.length;  iCnt++)
        {
            iAllQueue += processers[iCnt];
        }

        // Get System demand for each processor and for each of the classes
        sdWorkArray = new double[iAllQueue][totalWorkLoadClasses];

        int iNewIdx = processers.length;
        for (int iQueue=0; iQueue < processers.length; iQueue++)
        {
            for (int iClass=0; iClass < totalWorkLoadClasses; iClass++)
            {
                // MP resource calculation for original system demand =
                //      Original System Demand (SD) /  number of resource providers
                if (processers[iQueue] == 0)
                    sdWorkArray[iQueue][iClass] = 0;
                else
                    sdWorkArray[iQueue][iClass] =
                        sdArray[iQueue][iClass] / processers[iQueue];
            }

            if (processers[iQueue] > 1)
            {
                for (int iClass=0; iClass < totalWorkLoadClasses; iClass++)
                {
                    // MP resource for delay system demand =
                    //      Original SD * (number of resource providers - 1) / (number of
resource providers)
                    sdWorkArray[iNewIdx][iClass] =
                        sdArray[iQueue][iClass] * (processers[iQueue] - 1) / processers[iQueue];
                }
                iNewIdx++;
            }
        }
    }
}
```

9.

```
class AppWorkLoadModel extends RegistrationMBeanSupport
        implements WorkLoadModel /* , AppWorkLoadModelMBean */ {

    /**
 * Description:  WorkLoadModel for the application<br>
 */
    private static Category log = Category.getInstance(AppWorkLoadModel.class.getName());
    private ApplicationUC dcm;
    private int count = 0;
    private int updateCount = 0; // Every 2 count
    private ArrayList datalist;
    private Application application;
    private Collection clusters;
    private OutputGenerator appOutputGenerator;
    /**
     *  List of metric operations to be performed on arrival rates
     */
    private ArrayList outdesclist;
    boolean learningEnabled = true;
    // the MBeans server we are hosted in
    MBeanServer mbserver;
    // our object name
    ObjectName ourName;
    final ControllerMBeanHelper mbeanFactory = ControllerMBeanHelper.getInstance();
    /**
     * Construct a new Application Work Load model
     * @param application Application object in the DCM
     * @param appelm Element in the XML tree corresponding to the application parameters
     */
```

```
    public AppWorkLoadModel(Application application, Collection clusters, OutputGenerator outgen)
{
        // Generic initialization
        datalist = new ArrayList();
        outdesclist = new ArrayList();
        this.application = application;
        this.clusters = clusters;
        this.dcm = UCFactory.newApplicationUC();
        appOutputGenerator = outgen;
    }
    /**
     * Estimate the system responses given the arrival rate for the application.<br>
     * The algorithm being used is as follows:
     * <OL Type="1">
     * <LI>Enumerate through the required output list and determine if only app<br>
     * specific work load is needed.
     * <LI>If the required signal carries a metric operation class, perform the<br>
     * operation and return app output only.
     * <LI>If the signal does not carry a metric operation class, perform calculations<br>
     * on tier WorkLoadModel and find it from tier.
     * <LI>Perform the proper aggregation for the metric from all tier information.
     * </OL>
     */
    public void estimateResponse(TimeStampedValue[] arrivalRates,
                TimeStampedValueSet estimatedValues)
                throws WorkloadModelException {
        // Refresh the application object
        // Put in an array with processor size...
        int[] svrnum = new int[clusters.size()];
        Iterator cIter = clusters.iterator();
        for (int iTier = 0; cIter.hasNext(); iTier++) {
            Cluster cluster = (Cluster) cIter.next();
            svrnum[iTier] = dcm.findNumberOfServersByClusterId(cluster.getId());
            /*
             * The WLM cannot handle the situation where there are 0 servers
             * on one or more clusters, so we pretend that we have 1 server for now.
             */
            if (svrnum[iTier] == 0) {
               svrnum[iTier] = 1;
            }
        }
        try {
            appOutputGenerator.updateModelServers(svrnum);
            double[] pureArrivalRates = new double[arrivalRates.length];
            for (int iClass = 0; iClass < arrivalRates.length; iClass++) {
                pureArrivalRates[iClass] = arrivalRates[iClass].getValue();
            }
            // Retrieve the actual data
            double[][][] appResults =
                appOutputGenerator.generateResults(pureArrivalRates);
            ApplicationOutputDescriptor appOutDesc =
                appOutputGenerator.getApplicationOutputDescriptor();
            Iterator signalIter = appOutDesc.getOutputDescriptors().iterator();
            while (signalIter.hasNext()) {
                OutputDescriptor outDesc = (OutputDescriptor) signalIter.next();
                String signalType = outDesc.getSignalType();
                double clusterTotal = 0;
                cIter = clusters.iterator();
                for (int iTier = 0; cIter.hasNext(); iTier++) {
                    Cluster cluster = (Cluster) cIter.next();
                    /** @todo: Remove assumption of service class 0 */
                    double clusterResult = outDesc.getResult(iTier, 0);
                    clusterTotal += clusterResult;
                    estimatedValues.setClusterValue(
                        cluster.getId(),
                        signalType, clusterResult);
                    // Generate signals for servers in cluster
                    Collection servers = dcm.findServersByClusterId(cluster.getId());
                    Iterator serverIter = servers.iterator();
```

```
                    while (serverIter.hasNext()) {
                        Server currentServer = (Server) serverIter.next();
                        // Check how the server value is derived
                        double serverResult;
                        if (outDesc.getServerAggregate() ==
                                AggregatingFilter.AVERAGE) {
                            serverResult = outDesc.getResult(iTier, 0);
                        }
                        else { // if (outDesc.getServerAggregate() == Signal.AGGREGATION_SUMMARY)
{
                            serverResult = outDesc.getResult(iTier, 0) /
                                servers.size();;
                        }
                        estimatedValues.setServerValue(currentServer.getId() ,
                            signalType, serverResult);
                    }
                }
                // Check how the cluster value is derived
                double appResult;
                if (outDesc.getClusterAggregate() == AggregatingFilter.AVERAGE) {
                    appResult = clusterTotal / clusters.size();
                }
                else { // if (outDesc.getClusterAggregate() == Signal.AGGREGATION_SUMMARY) {
                    appResult = clusterTotal;
                }
                estimatedValues.setApplicationValue(appOutDesc.getApplicationId(),
                    signalType, appResult);
            }
        }
        catch (WorkloadModelException wlmex) {
            throw wlmex;
        }
    } //-- estimateResponse
/*
    // MBeanRegistration support (indirect)
    public ObjectName getObjectName( MBeanServer server, ObjectName name )
        throws MalformedObjectNameException {
        if (server != null) {
            this.mbserver = server;
            mbeanFactory.init(server);
        }
        if (name != null) {
            ourName = name;
        } else {
            ourName = new ObjectName(OBJECT_NAME);
        }
        return ourName;
    }
    public ObjectName getObjectName() {
        return ourName;
    }
    public void setObjectName( ObjectName oname ) {
        ourName = oname;
    }
*/
    // MBean implementations
    public int getUpdateCount() {
      return updateCount;
    }
    public void setUpdateCount(int newUpdateCount) {
      updateCount = newUpdateCount;
    }
    public boolean isLearningEnabled() {
        return learningEnabled;
    }
    //
    // New implementation, integrated with data acquisition engine
    //
    // train models using real data (used by application controller)
```

```
    public void trainOnline(TimeStampedValueSet sample) {
        // Update the Queuing model at regular intervals.
        if (count <= 0) {
            try {
                appOutputGenerator.trainOnline(sample);
                count = updateCount;
            }
            catch (WorkloadModelException wlme) {
                // Just log the problem for not able to train
                log.info(wlme.getLogString());
            }
        } else {
            count--;
        }
    }
}
```

10.
```
class ANOVAPredictor implements Predictor, ANOVAPredictorMBean {
    private static Category log = Category.getInstance(ANOVAPredictor.class.getName());

    private WeeklyANOVA predictionModel = null;
    private Application application = null;
    private ApplicationUC dcm; // The data center view
    private SignalDescriptor dcmSignalDescriptor = null;

    private DateHelper dateHelper = null;

    private final int TIME_BETWEEN_VALUES = 120000; // Milliseconds
    private final int MAX_INPUT_DATA_GAP = 10000; // Number of values that may be missing from
history or current data

    String historySynchronizeObject = "";

    boolean learningEnabled = true;

    public ANOVAPredictor(ApplicationUC dcm) {
        dateHelper = new DateHelper(Calendar.MONDAY);
//        dcm = UCFactory.newApplicationUC();
        this.dcm = dcm;
    } //- Ctor

    public void init(Application application)
            throws PredictorException {
        predictionModel = new WeeklyANOVA(TIME_BETWEEN_VALUES, MAX_INPUT_DATA_GAP);

        this.application = application;
        int appId = application.getId();

        //-- Set the history data for prediction
        synchronized (historySynchronizeObject) {
            dcmSignalDescriptor = dcm.findSignalDescriptor(
                    appId, SignalType.ARRIVAL_RATE.getName());

            if (dcmSignalDescriptor == null) {
                log.warn("SignalDescriptor for Application ID="+
                    appId+" signal="+SignalType.ARRIVAL_RATE.getName()+
                    " was not found so let's create one");

                try {
                    dcm.createSignalDescriptor(appId, SignalType.ARRIVAL_RATE.getName());
                }
                catch (DataCenterException de) {
                    throw new PredictorException(ErrorCode.cannotCreateSignalDescriptor,
                        application.getName(), de);
                }
                dcmSignalDescriptor = dcm.findSignalDescriptor(
                    appId, SignalType.ARRIVAL_RATE.getName());

                if (dcmSignalDescriptor == null) {
```

```
                    throw new PredictorException(ErrorCode.cannotCreateSignalDescriptor,
                        application.getName());
                }
            }

            Period historyPeriod = getHistoryBestPeriod();
            if ((historyPeriod.getStart() != null) &&
                (historyPeriod.getEnd() != null)) {
                TimeStampedValue[] history =
                     dcm.findSignalSamples(dcmSignalDescriptor.getId(),
                         historyPeriod.getStart(), historyPeriod.getEnd());
                predictionModel.setHistoryValues(history, true);
            } else {
                log.info("The history for prediction was not set for application '" +
application.getName() + "'.");
            }
        }

        //-- Set the current data for prediction
        long currentWeekStartTime = dateHelper.getWeekStart(new Date());
        TimeStampedValue[] currentValues = dcm.findSignalSamples(
                        dcmSignalDescriptor.getId(),
                        new Date(currentWeekStartTime),
                        new Date(currentWeekStartTime + DateHelper.MILLISECONDS_PER_WEEK));
        predictionModel.setCurrentValues(currentValues);
    }

    public void trainOnline(TimeStampedValue arrivalRate)
            throws PredictorException {
        if ((arrivalRate != null) && arrivalRate.isAvailable()) {
            TimeStampedValue[] predictorNewValues = predictionModel.setCurrentValue(arrivalRate);

            //-- Save new values to the database.
            if ((dcmSignalDescriptor != null) && (predictorNewValues != null)) {
                try {
                    dcm.storeSignalSamples(
                            dcmSignalDescriptor.getId(), predictorNewValues);
                } catch (DataCenterException ex) {
                    StringBuffer buf = new StringBuffer();
                    buf.append("Error calling dcm.storeSignalSamples().");
                    buf.append(" dcmSignalDescriptor.getId()="+dcmSignalDescriptor.getId());
                    for (int i=0; i<predictorNewValues.length; i++) {
                        buf.append("
predictorNewValues["+i+"]="+predictorNewValues[i].getValue());
                    }

                    log.error (buf.toString());
                    throw new PredictorException(ErrorCode.cannotStoreSignalValues,
application.getName(), ex);
                }
            }
        }
    } //-- trainOnline

    public TimeStampedValue[] predictArrivalRates(long time)
            throws PredictorException {

        double predictedValue = 0;
        Date predictionDate = new Date(time);
        synchronized (historySynchronizeObject) {
            predictedValue = predictionModel.predict(predictionDate);
            TimeStampedValue[] result = new TimeStampedValue[1];
            result[0] = new TimeStampedValue(predictionDate, predictedValue);
            return result;
        }
    } //-- predictArrivalRates

    private Period getHistoryBestPeriod() {
        Period historyBestPeriod = new Period(dcmSignalDescriptor.getBestPeriodStart(),
dcmSignalDescriptor.getBestPeriodEnd());
```

```
        if ((historyBestPeriod.getStart() == null) ||
            (historyBestPeriod.getEnd() == null)) {
            Period historyPeriod = dcm.findHistoryPeriod(dcmSignalDescriptor.getId());

            if (historyPeriod.getEnd() != null) {
                long weekStart = Math.max(historyPeriod.getStart().getTime(),
                                          dateHelper.getWeekStart(historyPeriod.getEnd()));
                boolean bFoundHistoryPeriod = false;
                if ((historyPeriod.getEnd().getTime() - weekStart) >=
(DateHelper.MILLISECONDS_PER_WEEK - TIME_BETWEEN_VALUES)) {
                    dcmSignalDescriptor.setBestPeriodStart(new Date(weekStart));
                    dcmSignalDescriptor.setBestPeriodEnd(historyPeriod.getEnd());
                    bFoundHistoryPeriod = true;
                } else if (weekStart >= (historyPeriod.getStart().getTime() +
DateHelper.MILLISECONDS_PER_WEEK)) {
                    weekStart -= DateHelper.MILLISECONDS_PER_WEEK;
                    dcmSignalDescriptor.setBestPeriodStart(new Date(weekStart));
                    dcmSignalDescriptor.setBestPeriodEnd(new Date(weekStart +
DateHelper.MILLISECONDS_PER_WEEK));
                    bFoundHistoryPeriod = true;
                }
                if (bFoundHistoryPeriod) {
                    try {
                        historyBestPeriod = new Period(dcmSignalDescriptor.getBestPeriodStart(),
                                                       dcmSignalDescriptor.getBestPeriodEnd());
                        dcm.storeHistoryBestPeriod(dcmSignalDescriptor.getId(),
                                historyBestPeriod);
                    } catch (DataCenterException ex) {
                        log.error(ex.getLogString());
                    }
                }
            }
        }

        return historyBestPeriod;
    } //-- getHistoryBestPeriod

    public Period trainPrediction(Period history, long timeAhead)
            throws PredictorException {
        if (dcmSignalDescriptor == null) {
            throw new PredictorSystemException(ErrorCode.uninitializedPredictorInstance,
"trainPrediction: Null signal descriptor.");
        }

        try {
            dcm.storeHistoryLastTrained(dcmSignalDescriptor.getId(), new Date());
        } catch (Throwable ex) {
            throw new PredictorException(ErrorCode.unexpectedPredictorError, ex);
        }

        Period result = null;

        int stepsAhead = (int) (timeAhead / TIME_BETWEEN_VALUES);
        if (stepsAhead <= 0) {
            return null;
        }

        Period wholeHistoryPeriod = dcm.findHistoryPeriod(dcmSignalDescriptor.getId());
        if ((wholeHistoryPeriod.getStart() == null) || (wholeHistoryPeriod.getEnd() == null)) {
            //-- We do not actually have history data...
            return null;
        }

        //-- Adjust the train period to the actual period for which we have data
        long trainStartTime = 0;
        if (history.getStart() != null) {
            trainStartTime = history.getStart().getTime();
        }
        trainStartTime = Math.max(trainStartTime, wholeHistoryPeriod.getStart().getTime());
        long trainEndTime = (new Date()).getTime();
```

```
        if (history.getEnd() != null) {
            trainEndTime = history.getEnd().getTime();
        }
        trainEndTime = Math.min(trainEndTime, wholeHistoryPeriod.getEnd().getTime());

        WeeklyANOVA tempPredictionModel = new WeeklyANOVA(TIME_BETWEEN_VALUES,
MAX_INPUT_DATA_GAP);
        long weekStartTime = dateHelper.getWeekStart(new Date(trainStartTime));
        if (weekStartTime < trainStartTime) {
            weekStartTime += DateHelper.MILLISECONDS_PER_WEEK;
        }

        double bestMatchPercentage = 0;
        //-- Build a prediction model for each week in the history.
        //   Apply this model to each following week in the history and
        //   measure the prediction matching.
        while (((weekStartTime + DateHelper.MILLISECONDS_PER_WEEK) <= trainEndTime) &&
                ((weekStartTime + DateHelper.MILLISECONDS_PER_WEEK * 2) <=
history.getEnd().getTime())) {
            //-- Get the history values that will be used to build the prediction model
            long weekEndTime = weekStartTime + DateHelper.MILLISECONDS_PER_WEEK -
TIME_BETWEEN_VALUES;
            TimeStampedValue[] historyData = dcm.findSignalSamples(dcmSignalDescriptor.getId(),
                                                    new Date(weekStartTime), new Date(weekEndTime));
            try {
                //-- Build/calculate the prediction model
                tempPredictionModel.setHistoryValues(historyData, false);

                if (tempPredictionModel.isReady()) {
                    long currentWeekStartTime = weekStartTime + DateHelper.MILLISECONDS_PER_WEEK;
                    long currentWeekEndTime = currentWeekStartTime +
DateHelper.MILLISECONDS_PER_WEEK - TIME_BETWEEN_VALUES;
                    double currentMatchPercentage = 0;
                    int numberOfWeeks = 0;

                    while (currentWeekEndTime <= history.getEnd().getTime()) {
                        historyData = dcm.findSignalSamples(dcmSignalDescriptor.getId(),
                                                            new Date(currentWeekStartTime), new
Date(currentWeekEndTime));
                        double currentWeekMatchPercentage = 0;
                        int numberOfEvaluations = 0;

                        double[] currentHistory =
tempPredictionModel.normalizeInputValues(historyData);

                        //-- For each history value, predict and measure the matching with the
real value
                        long currentTime = currentWeekStartTime;
                        for (int iStep = 0; iStep < currentHistory.length - stepsAhead; ++iStep)
{
                            tempPredictionModel.setCurrentValue(new TimeStampedValue(new
Date(currentTime), currentHistory[iStep]));
                            long predictTime = currentTime + stepsAhead * TIME_BETWEEN_VALUES;
                            double predictedValue = tempPredictionModel.predict(new
Date(predictTime));
                            currentWeekMatchPercentage +=
                                (1 - (Math.abs(predictedValue - currentHistory[iStep +
stepsAhead]) /
                                        currentHistory[iStep + stepsAhead]));

                            currentTime += TIME_BETWEEN_VALUES;
                        }
                        currentWeekMatchPercentage /= (currentHistory.length - stepsAhead);
                        currentMatchPercentage += currentWeekMatchPercentage;
                        ++numberOfWeeks;

                        //-- Go to the next week
                        currentWeekStartTime += DateHelper.MILLISECONDS_PER_WEEK;
                        currentWeekEndTime += DateHelper.MILLISECONDS_PER_WEEK;
                    }
```

```
                    if (numberOfWeeks > 0) {
                        currentMatchPercentage /= numberOfWeeks;
                        if (bestMatchPercentage < currentMatchPercentage) {
                            bestMatchPercentage = currentMatchPercentage;
                            result = new Period(new Date(weekStartTime), new Date(weekEndTime));
                            log.debug("trainPrediction: New best matching=" +
(bestMatchPercentage * 100) + "%, Period:" + result);
                        }
                    }
                }
            } catch (PredictorException ex) {
                //-- Ignore it
            } finally {
                weekStartTime += DateHelper.MILLISECONDS_PER_WEEK;
            }
        }

        if (result != null) {
            boolean replaceBestHistoryPeriod = false;

            if ((dcmSignalDescriptor.getBestPeriodStart() != null) &&
                (dcmSignalDescriptor.getBestPeriodEnd() != null)) {
                if ((dcmSignalDescriptor.getBestPeriodStart().getTime() !=
result.getStart().getTime()) ||
                    (dcmSignalDescriptor.getBestPeriodEnd().getTime() !=
result.getEnd().getTime())) {
                    replaceBestHistoryPeriod = true;
                }
            } else {
                replaceBestHistoryPeriod = true;
            }

            if (replaceBestHistoryPeriod) {
                synchronized (historySynchronizeObject) {
                    dcmSignalDescriptor.setBestPeriodStart(result.getStart());
                    dcmSignalDescriptor.setBestPeriodEnd(result.getEnd());
                    try {
                        dcm.storeHistoryBestPeriod(dcmSignalDescriptor.getId(),
                                new Period(dcmSignalDescriptor.getBestPeriodStart(),
                                    dcmSignalDescriptor.getBestPeriodEnd()));
                    } catch (DataCenterException ex) {
                        throw new PredictorException(ErrorCode.unexpectedPredictorError, ex);
                    }
                    init(application);
                }
            }
        }

        return result;
    } //-- trainPrediction

    public double getErrorVariance(long now, long time)
            throws PredictorException {
        return predictionModel.getErrorVariance(now, time);
    } //-- getErrorVariance

    public boolean isReady() {
        if (predictionModel == null) {
            return false;
        }
        return predictionModel.isReady();
    }

    public Date getLastPredictionTrainingTime() {
        if (dcmSignalDescriptor == null) {
            return null;
        }
        return dcmSignalDescriptor.getLastTrained();
    }
```

```
    // MBean related methods go below
    public boolean isLearningEnabled() {
        return learningEnabled;
    }
}
```

11.
```
public class WeeklyANOVA {
/**
 * Description:  Prediction algorithm using the ANOVA method on a one week worth of history
data<br>
 */

    private static Category log = Category.getInstance(WeeklyANOVA.class.getName());

    private int autoregressionDegree = 2;

    public final static int WEEK_SIZE = 7;
    /**
     * The time between each two history or current values (in milliseconds)
     */
    private int timeBetweenSamples = 1000;
    private long historyStartTime = 0;
    private long historyEndTime = 0;

    private int samplesPerDay = 0;
    private double weekAverage = 0;
    private int maxGapToFill = 2;

    ANOVAModel workdaysModel = null;
    ANOVAModel weekendModel = null;

    private double[] historyValues = null;
    private double[] currentValues = null;

    int lastCurrentIndex = -1;
    private TimeStampedValue lastCurrentValue = new TimeStampedValue(new Date(), 0);
    DateHelper dateHelper = null;

    private long calculateDuration = 0;
    private long createDuration = 0;

    private boolean wasHistoryLoaded = false;
    private boolean isReady = false;

    /**
     * Constructor
     *
     * @param initTimeBetweenSamples The time interval between any two history or current values.
     * @param initMaxGapToFill The max data gap (history or current) that will be filled
automatically.
     * @param initPersistCurrentData Whether the current data set using
<code>setRealValue()</code>
     *        must be persisted or not.
     */
    public WeeklyANOVA(
            int initTimeBetweenSamples,
            int initMaxGapToFill) {

        dateHelper = new DateHelper(Calendar.MONDAY);

        timeBetweenSamples = initTimeBetweenSamples;
        maxGapToFill = initMaxGapToFill;
        samplesPerDay = Math.round(DateHelper.MILLISECONDS_PER_DAY / timeBetweenSamples);

        historyStartTime = dateHelper.getWeekStart(new Date());
        historyEndTime = historyStartTime + DateHelper.MILLISECONDS_PER_WEEK;

        //-- Allocate memory for current values
        currentValues = new double[WEEK_SIZE * samplesPerDay];
```

```
        //-- Set to zero all current values
        for (int i = 0; i < WEEK_SIZE * samplesPerDay; ++i) {
            currentValues[i] = 0;
        }

        //-- Initialize ANOVA models for work days and weekend
        workdaysModel = new ANOVAModel(0, 4, samplesPerDay, autoregressionDegree);
        weekendModel = new ANOVAModel(5, 6, samplesPerDay, autoregressionDegree);
    } //-- Ctor

    private int getIndexFromTime(long time, long weekStartTime) {
        return (int) ((time - weekStartTime) / timeBetweenSamples);
    }

    private int getHistoryIndexFromTime(long time) {
        return getIndexFromTime(time, historyStartTime);
    }
    private int getHistoryIndexFromTime(Date time) {
        return getHistoryIndexFromTime(time.getTime());
    }

    private int getCurrentIndexFromTime(long time) {
        long result = (time - dateHelper.getWeekStart(time)) / timeBetweenSamples;
        return (int) (result - (result / (WEEK_SIZE * samplesPerDay)) * (WEEK_SIZE *
samplesPerDay));
    }
    private int getCurrentIndexFromTime(Date time) {
        return getCurrentIndexFromTime(time.getTime());
    }

    public void setHistoryValues(TimeStampedValue[] initHistoryValues, boolean
isHistoryNormalized)
            throws PredictorException {

        long _startTime = System.currentTimeMillis();

        if ((initHistoryValues == null) || (initHistoryValues.length == 0)) {
            throw new PredictorException(ErrorCode.notEnoughData);
        }

        historyStartTime = dateHelper.getWeekStart(initHistoryValues[0].getTimeStamp());
        historyEndTime = historyStartTime + DateHelper.MILLISECONDS_PER_WEEK;

        //-- Initialize the history values
        if (isHistoryNormalized &&
            (initHistoryValues.length != WEEK_SIZE * samplesPerDay) &&
            (initHistoryValues.length != WEEK_SIZE * samplesPerDay + 1)) {
            isHistoryNormalized = false;
        }
        if (isHistoryNormalized) {
            //-- Allocate memory for history values
            historyValues = new double[WEEK_SIZE * samplesPerDay];

            for (int iSample = 0; iSample < historyValues.length; ++iSample) {
                historyValues[iSample] = initHistoryValues[iSample].getValue();
            }
        } else {
            historyValues = normalizeInputValues(initHistoryValues);
        }

        calculateModels();

        wasHistoryLoaded = true;

        createDuration = System.currentTimeMillis() - _startTime;
        log.debug("WeeklyANOVA setHistoryValues finished in " + createDuration + "
milliseconds.");
    } //-- setHistoryValues(TimeStampedValue[]

    public void setHistoryValues(double[] initHistoryValues)
```

```
            throws PredictorException {
        if (initHistoryValues.length < WEEK_SIZE * samplesPerDay) {
            throw new PredictorException(ErrorCode.notEnoughData);
        }

        long _startTime = System.currentTimeMillis();

        historyStartTime = dateHelper.getWeekStart(new Date(System.currentTimeMillis() -
DateHelper.MILLISECONDS_PER_WEEK));
        historyEndTime = historyStartTime + DateHelper.MILLISECONDS_PER_WEEK;

        //-- Allocate memory for history values
        historyValues = new double[WEEK_SIZE * samplesPerDay];

        //-- Initialize the history values
        for (int iSample = 0; iSample < initHistoryValues.length; ++iSample) {
            historyValues[iSample] = initHistoryValues[iSample];
        }

        calculateModels();

        wasHistoryLoaded = true;

        createDuration = System.currentTimeMillis() - _startTime;
        log.debug("WeeklyANOVA setHistoryValues finished in " + createDuration + "
milliseconds.");
    } //-- setHistoryValues(double[]

    public double predict(Date time)
            throws PredictorException {
        if (wasHistoryLoaded == false) {
            if (lastCurrentIndex >= 0) {
                //return currentValues[lastCurrentIndex];
                if (lastCurrentValue == null) {
                    return (0.0);
                }
                else {
                    return lastCurrentValue.getValue();
                }
            } else {
                throw new PredictorException(ErrorCode.notEnoughData);
            }
        }

        int predictIndex = getCurrentIndexFromTime(time);

        return getModel(predictIndex).predict(predictIndex, currentValues, lastCurrentIndex);
    } //-- predict

    public void calculateModels()
            throws PredictorException {
        long _startTime = System.currentTimeMillis();
        workdaysModel.calculateModel(historyValues);
        isReady = workdaysModel.isReady();
        weekendModel.calculateModel(historyValues);
        isReady = isReady && weekendModel.isReady();
        for (int i = 0; i < autoregressionDegree; ++i) {
            currentValues[i] = historyValues[i];
        }
        calculateDuration = System.currentTimeMillis() - _startTime;
        log.debug("WeeklyANOVA.calculateModels finished in " + calculateDuration + "
milliseconds.");
    } //-- calculateModels

    /**
     * Fills history gaps by averaging the values at the start and at the end of the gap.
     *
     * The start and end indexes are not included in the gap.
     */
    private void fillHistoryGap(int startIndex, int endIndex)throws PredictorException {
```

```
        fillGap(historyValues, startIndex, endIndex);
    } //-- fillHistoryGap

    /**
     * Fills data gaps by averaging the values at the start and at the end of the gap.
     *
     * The start and end indexes are not included in the gap.
     */
    private void fillGap(double[] values, int startIndex, int endIndex)
            throws PredictorException {
        if (endIndex < startIndex) {
            startIndex = 0;
        }

        int gapSize = endIndex - startIndex - 1;

        if (gapSize <= 0) {
            //-- There's no gap. Nothing to do.
            return;
        } else if (gapSize > maxGapToFill) {
            //-- This gap is too big.
            throw new PredictorException(ErrorCode.notEnoughData);
        }

        double startValue = 0;
        if (startIndex >= 0) {
            startValue = values[startIndex];
        }
        double endValue = 0;
        if (endIndex < samplesPerDay) {
            endValue = values[endIndex];
        }
        double valueIncrement = (endValue - startValue) / gapSize;

        double currentValue = startValue;
        for (int iValue = startIndex + 1; iValue < endIndex; ++iValue) {
            currentValue += valueIncrement;
            values[iValue] = currentValue;
        }
    } //-- fillGap

    /**
     * Sets one real data sample in the matrix of current values
     *
     * @param newValue The value to be set
     *
     * @return The actual value used by the predictor (the timestamp may be
     *          different from the one in the <code>newValue</code>).<br>
     *          Returns null if <code>newValue</code> isn't used by the predictor.
     */
    public TimeStampedValue[] setCurrentValue(TimeStampedValue newValue)
            throws PredictorException {
        TimeStampedValue[] result = null;
        int stepIndex = getCurrentIndexFromTime(newValue.getTimeStamp());
        if (stepIndex == lastCurrentIndex) {
            //-- This value doesn't generate a new step
            //-- Store the new value as last known value and do nothing
            lastCurrentValue = newValue;
            return null;
        }

        currentValues[stepIndex] = newValue.getValue();

        if (stepIndex < lastCurrentIndex) {
            lastCurrentIndex = -1;
        }

        DateHelper localDateHelper = new DateHelper(Calendar.MONDAY);
        long currentWeekStartTime = localDateHelper.getWeekStart(newValue.getTimeStamp());
        int lastIndexForResult = lastCurrentIndex + 1;
```

```
        try {
            fillGap(currentValues, lastCurrentIndex, stepIndex);
        } catch (PredictorException ex) {
            if (ex.getErrorCode() == ErrorCode.notEnoughData) {
                lastIndexForResult = stepIndex;
            } else {
                throw ex;
            }
        }

        result = new TimeStampedValue[stepIndex - lastIndexForResult + 1];
        for (int iStep = lastIndexForResult; iStep <= stepIndex; ++iStep) {
            long normalizedTime = iStep * timeBetweenSamples + currentWeekStartTime;
            result[iStep - lastIndexForResult] = new TimeStampedValue(new Date(normalizedTime),
newValue.getValue());
        }

        lastCurrentIndex = stepIndex;

        if (result.length <= 10) {
            return result;
        } else {
            TimeStampedValue[] shortResult = new TimeStampedValue[10];
            for (int iSample = 0; iSample < 10; ++iSample) {
                shortResult[iSample] = result[result.length - 10 + iSample];
            }
            return shortResult;
        }
    } //-- setCurrentValue

    public void setCurrentValues(TimeStampedValue[] initCurrentValues)
            throws PredictorException {

        if ((initCurrentValues == null) || (initCurrentValues.length == 0)) {
            //-- Do nothing...
            return;
        }

        for (int iSample = 0; iSample < initCurrentValues.length; ++iSample) {
            lastCurrentIndex =
getCurrentIndexFromTime(initCurrentValues[iSample].getTimeStamp());
            currentValues[lastCurrentIndex] = initCurrentValues[iSample].getValue();
        }
    } //-- setCurrentValues

    /**
     * Returns the processing time consumed for the calculation of the model(s) (in milliseconds)
     */
    public long getCalculateDuration() {
        return calculateDuration;
    } //-- getCalculateDuration

    /**
     * Returns the processing time consumed for the creation of this object in milliseconds
     */
    public long getCreateDuration() {
        return createDuration;
    } //-- getCreateDuration


    public TimeStampedValue[] getHistoryData() {
        if (wasHistoryLoaded == false) {
            return null;
        }

        TimeStampedValue[] result = new TimeStampedValue[historyValues.length];
        long currentTime = historyStartTime;
        for (int iValue = 0; iValue < historyValues.length; ++iValue) {
            result[iValue] = new TimeStampedValue(new Date(currentTime), historyValues[iValue]);
            currentTime += timeBetweenSamples;
```

```
        }

        return result;
    } //-- getHistoryData

    public double[] normalizeInputValues(TimeStampedValue[] inputValues)
            throws PredictorException {

        if ((inputValues == null) || (inputValues.length == 0)) {
            throw new PredictorException(ErrorCode.notEnoughData);
        }

        //-- Allocate memory for result
        double[] result = new double[WEEK_SIZE * samplesPerDay];

        long weekStartTime = dateHelper.getWeekStart(inputValues[0].getTimeStamp());
        long weekEndTime = weekStartTime + DateHelper.MILLISECONDS_PER_WEEK - timeBetweenSamples;
        int lastValueIndex = -1;
        int currentValueIndex = 0;
        for (int iSample = 0; iSample < inputValues.length; ++iSample) {
            Date date = inputValues[iSample].getTimeStamp();
            if (date.getTime() > weekEndTime) {
                //-- We have more data than we need
                break;
            }

            currentValueIndex = getIndexFromTime(date.getTime(), weekStartTime);
            if (currentValueIndex == lastValueIndex) {
                //-- This sample doesn't generate a new step
                continue;
            }
            result[currentValueIndex] = inputValues[iSample].getValue();

            fillGap(result, lastValueIndex, currentValueIndex);

            lastValueIndex = currentValueIndex;
        }
        //-- Fill the gap at the end of the result data
        fillGap(result, lastValueIndex, samplesPerDay * WEEK_SIZE);

        return result;
    } //-- normalizeInputValues

    public double getErrorVariance(long now, long ahead)
            throws PredictorException {
        int nowIndex = getCurrentIndexFromTime(now);
        int aheadIndex = getCurrentIndexFromTime(ahead);

        return getModel(aheadIndex).getErrorVariance(aheadIndex - nowIndex);
    } //-- getErrorVariance

    private ANOVAModel getModel(int index) {
        int dayOfWeek = index / samplesPerDay;

        if (dayOfWeek < 5) {
            return workdaysModel;
        } else {
            return weekendModel;
        }
    } //-- getModel

    /**
     * Returns true if this predictor is ready to predict (ie: if the history was set and the
model could be calculated)
     */
    public boolean isReady() {
        return isReady;
    }

    /**
```

```
     * Dumps this object into a file (used for testing only)
     */
    public void toFile(String fileName)
            throws PredictorException {
        workdaysModel.toFile("Work days:", fileName, false);
        weekendModel.toFile("Weekend:", fileName, true);

        FileWriter outFile;
        try {
            outFile = new FileWriter(fileName, true);
        } catch (Exception ex) {
            throw new PredictorException(ErrorCode.cannotOpenFile, ex);
        }

        try {
            outFile.write("**************************\r\n");
            outFile.write("History:\r\n");
            /*
            for (int iStep = 0; iStep < samplesPerDay; ++iStep) {
                for (int iDay = 0; iDay < WEEK_SIZE; ++iDay) {
                    outFile.write("" + historyValues[iDay][iStep] + " ");
                }
                outFile.write("\r\n");
            }*/
            for (int iDay = 0; iDay < WEEK_SIZE; ++iDay) {
                for (int iStep = 0; iStep < samplesPerDay; ++iStep) {
                    outFile.write("" + historyValues[iDay * samplesPerDay + iStep] + "\r\n");
                }
                //outFile.write("\r\n");
            }

            outFile.write("\r\nDifferences:\r\n");
            /*double[] values = new double[5 * samplesPerDay];
            for (int iDay = 0; iDay < 5; ++iDay) {
                for (int iStep = 0; iStep < samplesPerDay; ++iStep) {
                    values[iDay * samplesPerDay + iStep] = historyValues[iDay][iStep];
                }
            }*/
            long time = historyStartTime;
            for (int iDay = 0; iDay < WEEK_SIZE; ++iDay) {
                ANOVAModel model;
                int diffDays = 0;
                if (iDay < 5) {
                    model = workdaysModel;
                } else {
                    diffDays = 5;
                    model = weekendModel;
                }
                for (int iStep = 0; iStep < samplesPerDay; ++iStep) {
                    outFile.write("" + (predict(new Date(time))) + "\r\n");
                    time += timeBetweenSamples;
                }
            }

        } catch (Exception ex) {
        } finally {
            try {
                outFile.close();
            } catch (Exception ex) {
            }
        }
    } //-- toFile
}
```

12.

```
public class ANOVAModel {
    private static Category log = Category.getInstance(ANOVAModel.class.getName());

    /** The prediction horizon */
    private static final int MAX_PREDICTION_TIME = 2 * 60 * 3600 * 1000; // Milliseconds
```

```
    /** The max number of steps ahead that we can predict */
    private int maxPredictionSteps = 0;

    //-- Nonstationary signal components
    public double average = 0;
    public double[] dayAverages = null; // alpha factor, size = numberOfDays
    public double[] stepAverages = null; // beta factor, size = samplesPerDay
    //-- Stationary signal component
    private double[] residues = null;

    /**
     * The variance of the difference between real history data and the result of
     * applying this model to the same real history data.<br>
     * IE: If Y(t) is the signal used to calculate this model<br>
     *     and Yp(t) is the signal predicted by this model (one step ahead),<br>
     *     then err(t) = Yp(t) - Y(t) and errorVariance is the variance of err(t).<br>
     * errorVariance[0] is the variance at current time t<br>
     * errorVariance[h] is the variance at t + h<br>
     */
    private double[] errorVariance = null;
    private double[] errorVarianceCoefficients = null;
    public double[] autoregressionCoefficients = null;

    private int startDay = 0;
    private int endDay = 0;
    private int numberOfDays = 0;
    private int samplesPerDay = 0;
    private int timeBetweenSamples = 0; // Milliseconds
    private int autoregressionDegree = 0;
    private boolean canCalculateErrorVariance = false;

    /**
     * Constructor
     *
     * @param startDay The start day (inclusive)
     * @param endDay The end day (inclusive)
     * @param samplesPerDay The number of steps/samples for each day
     * @param autoregressionDegree The number of autoregression coefficients
     */
    public ANOVAModel(int startDay, int endDay, int samplesPerDay, int autoregressionDegree) {
        this.startDay = startDay;
        this.endDay = endDay;
        this.numberOfDays = endDay - startDay + 1;
        this.samplesPerDay = samplesPerDay;
        this.autoregressionDegree = autoregressionDegree;

        timeBetweenSamples = (int) (DateHelper.MILLISECONDS_PER_DAY / samplesPerDay);
        maxPredictionSteps = MAX_PREDICTION_TIME / timeBetweenSamples;

        dayAverages = new double[numberOfDays];
        stepAverages = new double[samplesPerDay];
        residues = new double[numberOfDays * samplesPerDay];

        errorVariance = new double[maxPredictionSteps + 1];

        cleanUp();
    } //-- Ctor

    private void cleanUp() {
        average = 0;

        for (int iDay = 0; iDay < numberOfDays; ++iDay) {
            dayAverages[iDay] = 0;
        }

        for (int iStep = 0; iStep < samplesPerDay; ++iStep) {
            stepAverages[iStep] = 0;
        }

        for (int i = 0; i < numberOfDays * samplesPerDay; ++i) {
```

```
            residues[i] = 0;
        }

        for (int i = 0; i <= maxPredictionSteps; ++i) {
            errorVariance[i] = 0;
        }

        autoregressionCoefficients = null;
        errorVarianceCoefficients = null;
        canCalculateErrorVariance = false;
    } //-- cleanUp

    /**
     * Calculates this model.
     *
     * @param samples Array of history values to use for calculation
     *
     * @return True if the successful.
     */
    public boolean calculateModel(double[] samples)
            throws PredictorException {
        cleanUp();
        double totalSum = 0;

        //-- Sum values per day, per step and overall
        int iDay = 0;
        int iStep = 0;
        for (int iValue = startDay * samplesPerDay; iValue < (endDay + 1) * samplesPerDay;
++iValue) {
            dayAverages[iDay] += samples[iValue];
            stepAverages[iStep] += samples[iValue];
            average += samples[iValue];

            ++iStep;
            if (iStep == samplesPerDay) {
                iStep = 0;
                ++iDay;
            }
        }

        //-- Calculate overall average
        average /= (samplesPerDay * (endDay - startDay + 1));

        //-- Calculate day averages (minus overall average)
        for (iDay = 0; iDay <= endDay - startDay; ++iDay) {
            dayAverages[iDay] = dayAverages[iDay] / samplesPerDay - average;
        }

        //-- Calculate step averages (minus overall average)
        for (iStep = 0; iStep < samplesPerDay; ++iStep) {
            stepAverages[iStep] = stepAverages[iStep] / (endDay - startDay + 1) - average;
        }

        //-- Calculate the residues
        for (iDay = 0; iDay <= endDay - startDay; ++iDay) {
            for (iStep = 0; iStep < samplesPerDay; ++iStep) {
                residues[iDay * samplesPerDay + iStep] =
                        samples[(iDay +startDay) * samplesPerDay + iStep] -
                        dayAverages[iDay] - stepAverages[iStep] - average;
            }
        }

        calculateAutoregressionCoefficients(autoregressionDegree);
        calculateErrorVariance(samples);

        return canCalculateErrorVariance;
    } //-- calculateModel

    /**
     * Calculates the autoregression coefficients using the Yule-Walker method.
```

```
     *
     * @param degree The order/degree of the autoregressive model
     */
    private void calculateAutoregressionCoefficients(int degree)
            throws PredictorException {

        int numberOfValues = numberOfDays * samplesPerDay;
        double[][] autoCorrelationMatrix = new double[degree][degree + 1];
/*
        double[] autoCorrelationCoefficients = new double[degree + 1];
        for (int k = 0; k <= degree; ++k) {
            for (int n = 0; n < numberOfValues - k; ++n) {
                autoCorrelationCoefficients[k] += (residues[n] * residues[n + k]);
            }
            autoCorrelationCoefficients[k] /= (numberOfValues - k);
        }

        for (int i = 0; i < degree; ++i) {
            for (int j = i; j < degree; ++j) {
                autoCorrelationMatrix[i][j] = autoCorrelationCoefficients[j - i];
                //-- The matrix is symmetric to the first diagonal
                autoCorrelationMatrix[j][i] = autoCorrelationMatrix[i][j];
            }
            autoCorrelationMatrix[i][degree] = autoCorrelationCoefficients[i + 1];
        }*/

        double[] autoCorrelationCoefficientsDown = new double[degree + 1];
        double[] autoCorrelationCoefficientsUp = new double[degree + 1];

        for (int k = 0; k <= degree; ++k) {
            for (int n = degree; n < numberOfValues - degree; ++n) {
                autoCorrelationCoefficientsDown[k] += (residues[n] * residues[n + k]);
            }
            autoCorrelationCoefficientsDown[k] /= (numberOfValues - 2 * degree - k);
        }
        for (int i = 0; i < degree; ++i) {
            for (int j = 0; j <= i; ++j) {
                autoCorrelationMatrix[i][j] = autoCorrelationCoefficientsDown[i - j];
            }
            autoCorrelationMatrix[i][degree] = - autoCorrelationCoefficientsDown[i + 1];
        }

        for (int k = 0; k >= -degree; --k) {
            for (int n = degree; n < numberOfValues - degree; ++n) {
                autoCorrelationCoefficientsUp[-k] += (residues[n] * residues[n + k]);
            }
            autoCorrelationCoefficientsUp[-k] /= (numberOfValues - 2 * degree - k);
        }
        for (int i = 0; i < degree; ++i) {
            for (int j = i + 1; j < degree; ++j) {
                autoCorrelationMatrix[i][j] = autoCorrelationCoefficientsUp[j - i];
            }
        }
        autoregressionCoefficients = solveLinearEquations(autoCorrelationMatrix, degree);
        for (int i = 0; i < autoregressionCoefficients.length; ++i) {
            autoregressionCoefficients[i] = - autoregressionCoefficients[i];
        }
    } //-- calculateAutoregressionCoefficients
/*
    private static void printMatrix(double[][] mat, int degree) {
        for (int i = 0; i < degree; ++i) {
            String line = "";
            for (int j = 0; j <= degree; ++j) {
                line += mat[i][j] + ",   ";
            }
            log.debug(line);
        }
    }
*/
```

```
    private void calculateErrorVariance(double[] samples) {
        //-- error[] is the difference between predicted and real values
        double[] error = new double[numberOfDays * samplesPerDay];
        int indexInWeek = startDay * samplesPerDay;
        for (int i = 1; i < error.length; ++i) {
            ++indexInWeek;
            error[i] = predict(indexInWeek, samples, indexInWeek - 1) - samples[indexInWeek];
        }
        error[0] = 0;

        //-- Calculate the average error
        double errorAverage = 0;
        for (int i = 0; i < error.length; ++i) {
            errorAverage += residues[i];
        }
        errorAverage /= error.length;

        //-- Calculate the variance of error at current time
        errorVariance[0] = 0;
        for (int i = 0; i < error.length; ++i) {
            errorVariance[0] += (error[i] - errorAverage) * (error[i] - errorAverage);
        }
        errorVariance[0] /= (error.length - 1);
        errorVariance[0] = Math.sqrt(errorVariance[0]);

        //-- Calculate the solutions for
autoregressionCoefficients[1]*x^2+autoregressionCoefficients[0]-1 = 0
        errorVarianceCoefficients = solveSecondDegreeEquation(autoregressionCoefficients[1],
autoregressionCoefficients[0], -1);
        if (errorVarianceCoefficients == null) {
            canCalculateErrorVariance = false;
            log.info("Predictor cannot calculate the variance of error.");
            return;
        }
        errorVarianceCoefficients[0] = 1 / errorVarianceCoefficients[0];
        errorVarianceCoefficients[1] = 1 / errorVarianceCoefficients[1];
        double tempMultiplyBy = errorVariance[0] / Math.abs(errorVarianceCoefficients[0] -
errorVarianceCoefficients[1]);
        double p0 = 1;
        double p1 = 1;
        double tempSum = 0;
        for (int i = 1; i < errorVariance.length; ++i) {
            p0 *= errorVarianceCoefficients[0];
            p1 *= errorVarianceCoefficients[1];
            double tempDiff = p0 - p1;
            tempSum += tempDiff * tempDiff;
            errorVariance[i] = Math.sqrt(tempSum) * tempMultiplyBy;
        }
        canCalculateErrorVariance = true;
    } //-- calculateVariance

    public double predict(int predictIndex, double[] currentSamples, int
lastValidCurrentSampleIndex) {
        if ((lastValidCurrentSampleIndex < (startDay * samplesPerDay + autoregressionDegree - 1))
||
            (predictIndex < (startDay * samplesPerDay + autoregressionDegree))) {
            //-- We don't have enough data to predict.
            //   Return the last real data value
            return currentSamples[lastValidCurrentSampleIndex];
        }
        if (predictIndex <= lastValidCurrentSampleIndex) {
            //-- We have real value, no need to predict
            return currentSamples[predictIndex];
        }

        int dayOfWeek, indexInDay;
        double sample = 0;
        int regressionIndex = 0;
        double[] predictedValues = new double[predictIndex - lastValidCurrentSampleIndex];
        for (int tempIndex = lastValidCurrentSampleIndex + 1; tempIndex <= predictIndex;
```

```
++tempIndex) {
            regressionIndex = tempIndex;
            dayOfWeek = tempIndex / samplesPerDay;
            indexInDay = tempIndex % samplesPerDay;
            double value = getNonstationaryComponent(dayOfWeek, indexInDay);
            for (int i = 0; i < autoregressionDegree; ++i) {
                --regressionIndex;
                int regressionDayOfWeek = regressionIndex / samplesPerDay;
                int regressionIndexInDay = regressionIndex % samplesPerDay;

                if (regressionIndex <= lastValidCurrentSampleIndex) {
                    sample = currentSamples[regressionIndex];
                } else {
                    sample = predictedValues[regressionIndex - lastValidCurrentSampleIndex - 1];
                }
                value += autoregressionCoefficients[i] *
                        (sample -
                         getNonstationaryComponent(regressionDayOfWeek, regressionIndexInDay));
            }

            predictedValues[tempIndex - lastValidCurrentSampleIndex - 1] = value;
        }

        return predictedValues[predictIndex - lastValidCurrentSampleIndex - 1];
    } //-- predict

    /**
     * Solves a second degree equation: ax^2 + bx + c = 0.<br>
     *
     * Returns an array of two doubles that are the solutions/roots,
     * or null if this equation doesn't have real solutions.
     */
    private static double[] solveSecondDegreeEquation(double a, double b, double c) {
        double[] result = new double[2];

        if (a == 0) {
            result[0] = result[1] = - c / b;
            return result;
        }

        double delta = b * b - 4 * a * c;
        if (delta < 0) {
            return null;
        }
        delta = Math.sqrt(delta);
        result[0] = (- b - delta) / (2 * a);
        result[1] = (- b + delta) / (2 * a);

        return result;
    } //-- solveSecondDegreeEquation

    /**
     * Solves a system of liniar equations using the Gauss method.
     *
     * @param matrix The matrix for the system to be solved (degree rows, degree+1 columns)
     * @param degree The order/degree of the system
     *
     * @return The array of solution values
     *
     * @throws PredictorException If the system doesn't have a unique solution (ie the system is
singular)
     */
    private static double[] solveLinearEquations(double[][] matrix, int degree)
            throws PredictorException {

        double[] result = new double[degree];
        for (int i = 0; i < degree; ++i) {
            /*
            !!! Do not switch rows. We need to preserve the order of the solutions.
            //-- Find the max absolute value under the main diagonal on column i
```

```
            int maxiRow = i;
            double max = Math.abs(matrix[i][i]);
            for (int iRow = i + 1; iRow < degree; ++iRow) {
                if (max < Math.abs(matrix[iRow][i])) {
                    maxiRow = iRow;
                    max = Math.abs(matrix[iRow][i]);
                }
            }

            if (maxiRow != i) {
                //-- Swap rows i and maxj
                for (int iCol = i; iCol <= degree; ++iCol) {
                    double temp = matrix[i][iCol];
                    matrix[i][iCol] = matrix[maxiRow][iCol];
                    matrix[maxiRow][iCol] = temp;
                }
            }
            */

            double pivot = matrix[i][i];
            if (pivot == 0.0) {
                throw new PredictorException(ErrorCode.singularSystemOfLinearEquations);
            }

            //-- Divide the current line by the pivot, so we matrix[i,i] = 1
            for (int iCol = i; iCol <= degree; ++iCol) {
                matrix[i][iCol] /= pivot;
            }
            //-- Zero out all values on column i under the main diagonal
            for (int iRow = i + 1; iRow < degree; ++iRow) {
                for (int iCol = i + 1; iCol <= degree; ++iCol) {
                    matrix[iRow][iCol] -= matrix[i][iCol] * matrix[iRow][i];
                }
                matrix[iRow][i] = 0;
            }
        }

        for (int iRow = degree - 1; iRow >= 0; --iRow) {
            result[iRow] = matrix[iRow][degree];
            for (int iCol = iRow + 1; iCol < degree; ++iCol) {
                result[iRow] -= matrix[iRow][iCol] * result[iCol];
            }
        }
        return result;
    } //-- solveLinearEquations

    public double getNonstationaryComponent(int dayOfWeek, int indexInDay) {
        return average + dayAverages[dayOfWeek - startDay] + stepAverages[indexInDay];
    } //-- getNonstationaryComponent

    public double getErrorVariance(int stepsAhead)
            throws PredictorException {
        if (canCalculateErrorVariance == false) {
            throw new PredictorException(ErrorCode.cannotCalculateErrorVariance);
        }
        if (stepsAhead >= errorVariance.length) {
            throw new PredictorException(ErrorCode.beyondPredictionHorizon);
        }

        return errorVariance[stepsAhead];
    } //-- getErrorVariance

    public boolean isReady() {
        return canCalculateErrorVariance;
    }

    /**
     * Dumps this object into a file (used for testing only)
     */
```

```
public void toFile(String title, String fileName, boolean append)
            throws PredictorException {
        FileWriter outFile = null;
        try {
            outFile = new FileWriter(fileName, append);
        } catch (Exception ex) {
            if (append) {
                throw new PredictorException(ErrorCode.cannotCreateFile, ex);
            } else {
                throw new PredictorException(ErrorCode.cannotOpenFile, ex);
            }
        }

        try {
            if (append) {
                outFile.write("\r\n\r\n");
            }
            outFile.write("**************************\r\n");
            outFile.write(title + "\r\n");
            outFile.write("Average=");
            outFile.write("" + average + "\r\n\r\n");
            outFile.write("Day averages:\r\n");
            for (int iDay = 0; iDay < dayAverages.length; ++iDay) {
                outFile.write("" + dayAverages[iDay] + "\r\n");
            }
            outFile.write("\r\nStep averages:\r\n");
            for (int iStep = 0; iStep < stepAverages.length; ++iStep) {
                outFile.write("" + stepAverages[iStep] + "\r\n");
            }
            outFile.write("\r\nErrors:\r\n");
            for (int iStep = 0; iStep < stepAverages.length; ++iStep) {
                for (int iDay = 0; iDay < dayAverages.length; ++iDay) {
                    outFile.write("" + residues[iDay * samplesPerDay + iStep] + " ");
                }
                outFile.write("\r\n");
            }

            outFile.write("\r\nAutoregression Coefficients:\r\n");
            for (int i = 0; i < autoregressionCoefficients.length; ++i) {
                outFile.write("" + autoregressionCoefficients[i] + "\r\n");
            }

            outFile.write("\r\nError Variance Coefficients:\r\n");
            for (int i = 0; i < errorVarianceCoefficients.length; ++i) {
                outFile.write("" + errorVarianceCoefficients[i] + "\r\n");
            }
        } catch (Exception ex) {
        } finally {
            try {
                outFile.close();
            } catch (Exception ex) {
            }
        }
    } //-- toFile
}
```

13.

```
public abstract class AbstractProcessor implements WorkLoadModelProcessor {

    static Category log = Category.getInstance(AbstractProcessor.class.getName());

    protected   int     procTemplateId;
    protected   ArrayList subscribedList = new ArrayList();
    protected   ArrayList coreSignalList = new ArrayList(); // Compulsory signal
    protected   int     totalTiers = 0;
    protected   int     totalWorkLoadClasses = 0;
    protected   int[]   clusterIdMap;
    protected   ArrayList histories[]; // Dimension of tier
    protected   int      maxHistorySize;
    private     double[] outputRatio;
```

```
    protected   int     processorId;
    protected   int[]   processorInfoId;

    protected class OutputSignal {

        private int outputIndex; // Stores the index of storage for the signal
        private SignalType signalType;

        public OutputSignal(int outputIndex, SignalType signalType) {
            setOutputIndex(outputIndex);
            setSignalType(signalType);
        }

        public void setOutputIndex(int outputIndex) {
            this.outputIndex = outputIndex;
        }
        public int getOutputIndex() {
            return outputIndex;
        }
        public void setSignalType(SignalType signalType) {
            this.signalType = signalType;
        }
        public SignalType getSignalType() {
            return signalType;
        }
    }
    public AbstractProcessor() {
    }
    public ProcessorInfo[] storeProcessorParameters() {
        ProcessorInfo[] piTiers = new ProcessorInfo[totalTiers];
        for (int iTier = 0; iTier < totalTiers; iTier++) {
            piTiers[iTier] = new ProcessorInfo(processorInfoId[iTier],
                processorId, clusterIdMap[iTier]);
            piTiers[iTier].setProperties(new Properties());

//            for (int iClass = 0; iClass < totalWorkLoadClasses; iClass++) {
/*
              Properties tierProp = new Properties();

              tierProp.setProperty("cluster_id",
                  String.valueOf(clusterIdMap[iTier]));
*/
//                hmStore[iTier].put(new Integer(iClass), tierProp);

//            }

//            log.info ("histories["+iTier+"].size="+histories[iTier].size());
        }

        return piTiers;
    }
    public void configureProcessor(int iprocTemplateId, int totalServiceClass,
            HashMap configProps[], int newMaxHistorySize) {
        // Initialize
        subscribedList.clear();
        coreSignalList.clear();
        procTemplateId = iprocTemplateId;
        totalWorkLoadClasses = totalServiceClass;
        totalTiers = configProps.length;
        clusterIdMap = new int[totalTiers];
        outputRatio = new double[totalTiers];
        processorInfoId = new int[totalTiers];
        processorId = (int) getClusterProperty(configProps[0],
            0, "processor_id", -1);
        for (int iTier = 0; iTier < totalTiers; iTier++) {
            // Determine processor id for application...
            processorInfoId[iTier] = (int) getClusterProperty(
                configProps[iTier], 0, "processor_info_id", -1);
            clusterIdMap[iTier] = (int) getClusterProperty(
                configProps[iTier], 0, "cluster_id", -1);
```

```
            // Note that the ratio applies to the current tier and thus
            // to retrive the current tier's arrival rate, use ratio in
            // the previous tier.  Moreover, class independent parameters
            // are stored within class 0.
            outputRatio[iTier] = getClusterProperty(configProps[iTier],
                0, "outputRatio", 1);
            if (iTier > 0) { // Accumulative ratio to application arrival rate
                outputRatio[iTier] *= outputRatio[iTier-1];
            }
        }
        maxHistorySize = newMaxHistorySize;
        histories = new ArrayList[totalTiers];
        for (int iTier = 0; iTier < totalTiers; iTier++) {
            histories[iTier] = new ArrayList(newMaxHistorySize);
        }
    }
    public void subscribeOutputCheckIndex(int outputIndex,
            SignalType trainingSignal,
            int nextIndex) throws WorkloadModelException {
        if (nextIndex <= outputIndex) {
            throw new WorkloadModelException(
                ErrorCode.wlmInvalidConfiguration,
                "Subscribed index " + outputIndex +
                " not valid for processor id " + getStrId());
        }
        // Check if this could replace core signal such that it is not doubly
        // stored..
        for (int iIdx=0; iIdx < coreSignalList.size(); iIdx++) {
            SignalType signalType = (SignalType) coreSignalList.get(iIdx);
            if (signalType == trainingSignal) {
                coreSignalList.remove(iIdx); // Remove redundant signal
                break;
            }
        }
        subscribedList.add(new OutputSignal(outputIndex, trainingSignal));
    }
    public double[][][] deriveOutput(double[] arrivalRates)
            throws WorkloadModelException {
        // Check subscribed list...
        if (subscribedList.size() == 0) {
            // Nothing to return for subscription
            throw new WorkloadModelException(ErrorCode.wlmInvalidConfiguration,
                "No signals subscribed from processor.");
        }
        // Arrival rates has to be the same size of class
        if (arrivalRates.length != totalWorkLoadClasses) {
            throw new WorkloadModelException(
                ErrorCode.wlmInvalidConfiguration,
                "Current arrival rate " + arrivalRates.length +
                " expected " + totalWorkLoadClasses);
        }
        double[][][] output =
            new double[totalTiers][totalWorkLoadClasses][subscribedList.size()];
        return output;
    }
    public void trainOnline(TimeStampedValueSet sample)
        throws WorkloadModelException {
        // Store values in order of [Time][Tier][wlclass][Signals]
            // Match the tier to the sample cluster
            // wlclass always 0
            // Time is just circular bufferX
            // Signal determined in subscription
        for (int iTier = 0; iTier < totalTiers; iTier++) {
            HistoryRecord histRecord = new HistoryRecord(totalWorkLoadClasses,
                subscribedList.size(), coreSignalList.size(),
                sample.getTimeStamp());
            // For each tier the server count is different.
            int serverCount = (int) sample.getClusterValue(
                clusterIdMap[iTier], SignalType.NUMBER_OF_SERVERS).getValue();
            // temporary workaround for bug 1536. If serverCount is
```

```
        // zero, make it one.
        serverCount = (serverCount == 0) ? 1 : serverCount;
        if (serverCount < 1) { // No need to train this if no servers
            WorkloadModelException wlme = new WorkloadModelException(
                ErrorCode.wlmInvalidConfiguration, "Cluster " +
                clusterIdMap[iTier] + " has only " + serverCount +
                " servers which is not at least 1 server");
            log.warn(wlme.getLogString());
            return;
        }
        histRecord.setServerCount(serverCount);
        log.debug("Server count for whole tier " + iTier + " is " +
            serverCount);
            int iClass = 0;
        // Find out arrival rate from sample
        TimeStampedValue tsv = sample.getClusterValue(clusterIdMap[iTier],
            SignalType.ARRIVAL_RATE);
        double arrvRate;
        if (tsv.isAvailable()) {
            arrvRate = tsv.getValue();
        }
        else { // Not subscribed, use pre-built values
            arrvRate = getTierArrivalRatio(iTier) * sample.getClusterValue(
                clusterIdMap[0], SignalType.ARRIVAL_RATE).getValue();
        }
        log.debug("Arrival rate for whole tier " + iTier + " is " +
            arrvRate);
        histRecord.setArrivalRate(iClass, arrvRate);
        // Need to resolve into the signal...
        Iterator signalIter = subscribedList.iterator();
        while (signalIter.hasNext()) {
            OutputSignal outSign = (OutputSignal) signalIter.next();
            double value =
                sample.getClusterValue(clusterIdMap[iTier],
                outSign.getSignalType()).getValue();
            histRecord.setHistoryData(iClass,
                findSignalIndex(outSign.getOutputIndex()), value);
            log.debug("Signal value for whole tier " + iTier +
                " with index " + outSign.getOutputIndex() +
                " is " + value);
        }
        // Add signals which is core but not subscribed...
        Iterator coreSignalIter = coreSignalList.iterator();
        int iCoreCount = 0;
        while (coreSignalIter.hasNext()) {
            SignalType signalType = (SignalType) coreSignalIter.next();
            double value =
                sample.getClusterValue(clusterIdMap[iTier],
                signalType).getValue();
            histRecord.setCoreData(iClass,
                iCoreCount++, value);
        }

        // Store the history into a fix size window
        while (histories[iTier].size() >= maxHistorySize) {
            histories[iTier].remove(0);
        }
        histories[iTier].add(histRecord);
    }
    // Update the model
    try {
        updateModel();
    }
    catch (WorkloadModelException wlme) {
        if (wlme.getErrorCode() == ErrorCode.wlmInsufficientHistory) {
            // This error is ok, just a notification history is not ready...
            return;
        }
        throw wlme; // Otherwise, rethrow error
    }
```

```
    }
    /**
     * @return Output index of signal
     */
    protected int findSignalIndex(int iOutputIndex) {
        Iterator signalIter = subscribedList.iterator();
        int iCheck = 0;
        while (signalIter.hasNext()) {
            OutputSignal outSign = (OutputSignal) signalIter.next();
            if (outSign.getOutputIndex() == iOutputIndex) {
                return iCheck;
            }
            iCheck++;
        }
        return -1;
    }
    /**
     * @param signalName Name of signal to be searched
     * @return Output index of core signals
     */
    protected int findCoreSignalIndex(SignalType signalType) {
        int coreIndex = -1;
        Iterator coreIter = coreSignalList.iterator();
        while (coreIter.hasNext()) {
            coreIndex++;
            SignalType coreSignal = (SignalType) coreIter.next();
            if (coreSignal == signalType) {
                break;
            }
        }
        return coreIndex;
    }
    /**
     * Retrieve the average arrival rate from the history window
     *
     * @parameter iTier  The index of tier for arrival rate of interest
     * @parameter iClass The class of arrival rate of interest
     * @return The average arrival rate of the history
     */
    protected double getAverageArrivalRate(int iTier, int iClass) {
        double avgResult = 0;
        if (histories[iTier].size() != 0) {
            Iterator histIter = histories[iTier].iterator();
            while (histIter.hasNext()) {
                HistoryRecord hrecord = (HistoryRecord) histIter.next();
                avgResult += hrecord.getArrivalRate(iClass);
            }
            avgResult /= histories[iTier].size();
        }
        return avgResult;
    }

    /**
     * Retrieve the average normalized arrival rate from the history window
     *
     * @parameter iTier  The index of tier for arrival rate of interest
     * @parameter iClass The class of arrival rate of interest
     * @return The average arrival rate of the history
     */
    protected double getNormalizedAverageArrivalRate(int iTier, int iClass) {
        double avgResult = 0;
        if (histories[iTier].size() != 0) {
            Iterator histIter = histories[iTier].iterator();
            while (histIter.hasNext()) {
                HistoryRecord hrecord = (HistoryRecord) histIter.next();
                avgResult += hrecord.getArrivalRate(iClass) /
                    hrecord.getServersCount();
            }
            avgResult /= histories[iTier].size();
        }
```

```
        return avgResult;
    }
    /**
     * Retrieve the maximum arrival rate from the history window
     *
     * @parameter iTier  The index of tier for arrival rate of interest
     * @parameter iClass The class of arrival rate of interest
     * @return The maximum arrival rate of the history
     */
    protected double getMaximumArrivalRate(int iTier, int iClass) {
        double maxResult = 0;
        if (histories[iTier].size() != 0) {
            Iterator histIter = histories[iTier].iterator();
            while (histIter.hasNext()) {
                HistoryRecord hrecord = (HistoryRecord) histIter.next();
                double tempValue = hrecord.getArrivalRate(iClass);
                if (tempValue > maxResult) {
                    maxResult = tempValue;
                }
            }
        }
        return maxResult;
    }

    /**
     * Retrieve the normalized maximum arrival rate from the history window
     *
     * @parameter iTier  The index of tier for arrival rate of interest
     * @parameter iClass The class of arrival rate of interest
     * @return The maximum arrival rate of the history
     */
    protected double getNormalizedMaximumArrivalRate(int iTier, int iClass) {
        double maxResult = 0;
        if (histories[iTier].size() != 0) {
            Iterator histIter = histories[iTier].iterator();
            while (histIter.hasNext()) {
                HistoryRecord hrecord = (HistoryRecord) histIter.next();
                double tempValue = hrecord.getArrivalRate(iClass) /
                    hrecord.getServersCount();
                if (tempValue > maxResult) {
                    maxResult = tempValue;
                }
            }
        }
        return maxResult;
    }
    /**
     * Retrieve the latest arrival rate from the history window
     *
     * @parameter iTier  The index of tier for arrival rate of interest
     * @parameter iClass The class of arrival rate of interest
     * @return The latest arrival rate of the history
     */
    protected double getLatestArrivalRate(int iTier, int iClass) {
        double lastResult = 0;
        if (histories[iTier].size() != 0) {
            HistoryRecord hrecord = (HistoryRecord)
                histories[iTier].get(histories[iTier].size()-1);
            lastResult = hrecord.getArrivalRate(iClass);
        }
        return lastResult;
    }
    /**
     * Retrieve the latest arrival rate from the history window
     * @parameter iTier  The index of tier for arrival rate of interest
     * @parameter iClass The class of arrival rate of interest
     * @return The latest arrival rate of the history
     */
    protected double getNormalizedLatestArrivalRate(int iTier, int iClass) {
        double lastResult = 0;
```

```
        if (histories[iTier].size() != 0) {
            HistoryRecord hrecord = (HistoryRecord)
                histories[iTier].get(histories[iTier].size()-1);
            lastResult = hrecord.getArrivalRate(iClass) /
                hrecord.getServersCount();
        }
        return lastResult;
    }
    /**
     * Looks up history data and then core value to return.
     * @param hist History array of tiers to be searched upon
     * @param iSignalIndex Index in record to looked up.  If -1, will lookup core name
     * @param iHistory Index of history record to be looked up.
     * @param iTier Index of tier
     * @param iClass Index of Work Load Class
     * @param signalName Name of signal if Index is -1
     * @return The history or cord value stored in the history record.
     */
    protected double getHistoryDataValue(ArrayList[] hist,
            int iSignalIndex, int iHistory,
            int iTier, int iClass, SignalType signType) {
        double outputValue = 0;
        if (iSignalIndex != -1) {
            outputValue = ((HistoryRecord)hist[iTier].get(iHistory)).
                getHistoryData(iClass, iSignalIndex);
        }
        else {
            outputValue = ((HistoryRecord)hist[iTier].get(iHistory)).
                getCoreData(iClass,
                findCoreSignalIndex(signType));
        }
        return outputValue;
    }
    public void updateModelServers(int[] servers)
            throws WorkloadModelException {
        if (servers.length != totalTiers) {
            throw new WorkloadModelException(ErrorCode.wlmInvalidInputValues,
                new IllegalArgumentException("updateModelServers expects " +
                totalTiers + " clusters but gets " + servers.length + " servers."));
        }
    }
    /**
     * Retrieve the standard proeprty from the processor
     * @param hashSet HashMap set with the property
     * @param iClass Work load class number
     * @param propertName Name of property which value is to be retrieved
     * @param defaultValue Value to return if not found
     */
    protected double getClusterProperty(HashMap hashSet,
        int iClass, String propertyName, double defaultValue) {
        double resultValue = defaultValue;
        Properties propSet = (Properties) hashSet.get(new Integer(iClass));
        String strValue = propSet.getProperty(propertyName);
        if (strValue != null && strValue.length() != 0) {
            try {
                resultValue = Double.parseDouble(strValue);
            }
            catch (NumberFormatException nfe) { // Warning about incorrect param
                log.warn("WLM Cannot interpret " + propertyName +
                    " with value " + strValue + ".  Using Default WLM values");
                resultValue = defaultValue;
            }
        }
        else {
            log.warn("WLM Processor property '" + propertyName +
                "' is missing.  Using Default WLM value " + defaultValue);
            resultValue = defaultValue;
            propSet.setProperty(propertyName, Double.toString(defaultValue));
        }
        return resultValue;
```

```
    }

    /**
     * Retrieve the ratio of tier arrival rate to application arrival rate
     *
     * @param iTier Index of tier of interest
     * @return ratio of tier arrival rate to application arrival rate
     */
    protected double getTierArrivalRatio(int iTier) {
        if (iTier == 0) {
            return (double)1;
        }
        else {
            return outputRatio[iTier-1];
        }
    }
}
```

14.
```
class AppControllerBean implements AppController,
    AppControllerBeanMBean, MBeanRegistration {

/**
 * Description:  AppControllerBean to estimate the server requirements for one application<br>
 */
    private static Category log = Category.getInstance(AppControllerBean.class.getName());
    private DateHelper dateHelper = null;
    private Application _application;
    private int _applicationId;
    private Collection _clusters;
    private int _webClusterId;
    private WorkLoadModel _wlm;
    private Predictor _predictor;
    private AppClassifier _classifier;
    private long[] _predictionIntervals; // Stores an array of milliseconds from start for
prediction to happen
    private InfrastructureNeeds[] _infrastructureNeeds = null;
    private TimeStampedValueSet _signalValues = null;
    private ApplicationUC _dataCenterModel; // The data center view
    private ServiceLevelObjectiveSet _serviceLevelObjectives;
    private boolean _wlmSloChanged = false;
    private ArrivalRateStatistics arrivalRateStats;
    private double _cpuSloTrapThreshold;
    private long _lastWLMTrainTime;
    private long _trainWLMInterval = 60000; // Default train for every minute
    private int _trainingCount = 0;
    // MBean related
    // the MBeans server we are hosted in
    private MBeanServer mbserver;
    // our object name
    private ObjectName ourName;
    // MBean factory
    private final ControllerMBeanHelper mbeanHelper = ControllerMBeanHelper.getInstance();
    // MBean related attributes
    private int samplingPeriod = 0;
    private int calculationPeriod = 0;
    /**
     * Constructor
     * @name The application name
     */
    public AppControllerBean(Application application) {
        dateHelper = new DateHelper(Calendar.MONDAY);
        _dataCenterModel = UCFactory.newApplicationUC();
        _application = application;
        _applicationId = application.getId();
        _clusters = _dataCenterModel.findClustersByApplicationId(application.getId());
        _serviceLevelObjectives = new ServiceLevelObjectiveSet (_dataCenterModel, _clusters);
        _serviceLevelObjectives.loadFromDCM();
        _lastWLMTrainTime = 0;
```

```
        String wlmTrainInterval = _dataCenterModel.getProperty(DcmObjectId.KANAHA,
            KanahaComponent.APPLICATION_CONTROLLER.getId(), "WLMTrainInterval", "60000");
        try {
            _trainWLMInterval = Long.parseLong(wlmTrainInterval);
        }
        catch (Exception exc) {
        }

                if (_trainWLMInterval <= 0) {
                        log.info("WLM training is disabled for App="+_applicationId);
                } else {
                        log.info("The WLM training period is "+_trainWLMInterval+" mSec for
App="+_applicationId);
                }

        double priority = _application.getPriority();
        _cpuSloTrapThreshold = fetchCpuSloTrapThreshold ((int)_application.getPriority());

        Iterator tiers = _clusters.iterator();
        _webClusterId = ((Cluster) tiers.next()).getId();
        try {
            _wlm = WorkLoadModelFactory.createNewAppWLM(_dataCenterModel,
                    application, _clusters);
        }
        catch (KanahaApplicationException kae) {
            log.error(kae.getLogString());
            // Cannot continue if work load model is not valid, quit here.
            throw new KanahaSystemException(kae.getErrorCode(), kae.getLogString());
        }
        _predictor  = PredictorFactory.getInstance(_dataCenterModel);
        try {
            _predictor.init(_application);
        } catch (PredictorException ex) {
            if (ex.getErrorCode() == ErrorCode.notEnoughData) {
                log.info("Cannot initialize predictor for application '" + _application.getName()
+ "': " + ex.getMessage());
            } else {
                log.error(ex.getLogString());
            }
        }
       _classifier = AppClassifierFactory.getInstance();
        _classifier.init(_application, _clusters, _predictor);

        // use a moving window of samples.
        arrivalRateStats = new ArrivalRateStatistics (50);
    }
    public void subscribeDataAcquisitionEngine(DataAcquisitionEngineHelper daeHelper) {
        //-- Subscribe to all needed input signals
        try {
            //-- get Arrival Rate from the WEB tier
            daeHelper.subscribeClusterSignal(_webClusterId, SignalType.ARRIVAL_RATE);
            daeHelper.subscribeClusterSignal(_webClusterId, SignalType.RAW_ARRIVAL_RATE);
            // get CPU Utilization and number of servers Signals from all Clusters.
            Iterator iter = _clusters.iterator();
            while (iter.hasNext()) {
                int clusterId = ((Cluster) iter.next()).getId();
                daeHelper.subscribeClusterSignal(clusterId,
                    SignalType.CPU_UTILIZATION);
                daeHelper.subscribeClusterSignal(clusterId,
                    SignalType.NUMBER_OF_SERVERS);
            }
        } catch (SubscriptionException ex) {
            log.error(ex.getLogString());
        }
    } //-- subscribeDataAcquisitionEngine

    public void unSubscribeDataAcquisitionEngine(DataAcquisitionEngineHelper daeHelper) {
        //-- Unsubscribe to all input signals
        try {
            //-- get Arrival Rate from the WEB tier
```

```
            daeHelper.unSubscribeClusterSignal(_webClusterId, SignalType.ARRIVAL_RATE);
            daeHelper.unSubscribeClusterSignal(_webClusterId, SignalType.RAW_ARRIVAL_RATE);
            // get CPU Utilization and number of servers Signals from all Clusters.
            Iterator iter = _clusters.iterator();
            while (iter.hasNext()) {
                int clusterId = ((Cluster) iter.next()).getId();
                daeHelper.unSubscribeClusterSignal(clusterId,
                    SignalType.CPU_UTILIZATION);
                daeHelper.unSubscribeClusterSignal(clusterId,
                    SignalType.NUMBER_OF_SERVERS);
            }
        } catch (SubscriptionException ex) {
            log.error(ex.getLogString());
        }
    } //-- subscribeDataAcquisitionEngine

    public InfrastructureNeeds[] estimateInfrastructureNeeds(TimeStampedValueSet signalValues)
            throws AppControllerException {

        //-- Train prediction (once a week)
        try {
            Date lastTrainPredictionTime = _predictor.getLastPredictionTrainingTime();
            boolean bTrainPrediction = false;
            if (lastTrainPredictionTime == null) {
                bTrainPrediction = true;
            } else if (lastTrainPredictionTime.getTime() + 10 * 60 * 1000 < (new
Date()).getTime()) {
                long thisWeekStart = dateHelper.getWeekStart(new Date());

                if (lastTrainPredictionTime.getTime() <= thisWeekStart) {
                    bTrainPrediction = true;
                }
            }
            if (bTrainPrediction) {
                long thisWeekStart = dateHelper.getWeekStart(new Date());
                Period trainPeriod = new Period(new Date(thisWeekStart - 90 *
DateHelper.MILLISECONDS_PER_DAY),
                                                new Date(thisWeekStart));
                trainPrediction(trainPeriod);
            }
        } catch (KanahaApplicationException ex) {
            log.error("AppControllerBean.estimateSimulatedInfrastructureNeeds trying to train
prediction:" + ex.getLogString());
        } catch (Exception ex) {
            String[] messageParams = new String[2];
            messageParams[0] = _application.getName();
            messageParams[1] = ex.getMessage();
            AppControllerException acEx = new
AppControllerException(ErrorCode.cannotTrainPrediction, messageParams, ex);
            log.error("AppControllerBean.estimateSimulatedInfrastructureNeeds trying to train
prediction:" + acEx.getLogString());
        }

        InfrastructureNeeds[] needs = new InfrastructureNeeds[_predictionIntervals.length + 1];
//prediction intervals plus current time

        // needs[0] represents the current needs for this app
        // For each prediction interval, predict the needs at that time
        for (int i = 0; i < _predictionIntervals.length + 1; i++) {
            long estimationTime = signalValues.getTimeStamp().getTime();
            if (i != 0) {
                estimationTime += _predictionIntervals[i - 1];
            }
            SignalValueSet estimatedValues = new SignalValueSet(new Date(estimationTime));
            TimeStampedValue[] prediction = null;
            try {
                prediction = _predictor.predictArrivalRates(estimationTime);
                _wlm.estimateResponse(prediction, estimatedValues);
            } catch (PredictorException ex) {
                throw new AppControllerException(ErrorCode.cannotEstimateInfrastructureNeeds,
```

```
_application.getName(), ex);
            } catch (WorkloadModelException wlmex) {
                ErrorCode errorCode = wlmex.getErrorCode();
                if (errorCode == ErrorCode.wlmInsufficientHistory) {
                    throw new AppControllerException(errorCode, wlmex);
                }
                else {
                    throw new AppControllerException(ErrorCode.cannotEstimateInfrastructureNeeds,
                        wlmex);
                }
            }
            /**
             * create a new SLO set and update it with the values calculated
             * by the WLM.
             * A new copy of the SLO set is required to guarantee that
             * the SLO values don't change while the ResourceBroker is calling
             * the BP functions.
             */
            if (i == 0) {
                    _serviceLevelObjectives = new ServiceLevelObjectiveSet (_dataCenterModel,
_clusters);
                    _serviceLevelObjectives.update(estimatedValues);
                    log.debug(_serviceLevelObjectives.toString());

                    StringBuffer buf = new StringBuffer();
                    buf.append ("webClusterId="+_webClusterId);
                    buf.append (" nServers="+
                        signalValues.getClusterValue(_webClusterId,
                        SignalType.NUMBER_OF_SERVERS.getName()).valueToString());
                    buf.append (" AR="+
                        signalValues.getClusterValue(_webClusterId,
                        SignalType.ARRIVAL_RATE.getName()).valueToString());
                    buf.append (" currCPU="+
                        signalValues.getClusterValue(_webClusterId,
                        SignalType.CPU_UTILIZATION.getName()).valueToString());
                    buf.append (" critCPU="+
                        estimatedValues.getClusterValue(_webClusterId,
                        SignalType.CRITICAL_CPU_UTILIZATION.getName()).valueToString());
                    buf.append (" critAR="+
                        estimatedValues.getClusterValue(_webClusterId,
                        SignalType.CRITICAL_ARRIVAL_RATE.getName()).valueToString());
                    log.info(buf.toString());
            }

            double totalArrivalRate = 0;
            for (int iWorkLoadClass = 0; iWorkLoadClass < prediction.length; ++iWorkLoadClass) {
                totalArrivalRate += prediction[iWorkLoadClass].getValue();
            }
            estimatedValues.setClusterValue(_webClusterId,
                    SignalType.ARRIVAL_RATE.getName(), totalArrivalRate);
            /*
            if (application.getId() == 1) {
                int kk = 0;
                for (int ii = 0; ii < systemResponse.length; ++ii) {
                    if (systemResponse[ii].getSignalDesc().getProperty("appid").equals("" +
application.getId())) {
                        String s = "" + ii + ":" +
systemResponse[ii].getSignalDesc().getSignalType().getName() +
                            " Value=" + systemResponse[ii].getData() +
                            " AggType=" +
systemResponse[ii].getSignalDesc().getSignalType().getAggType().toString() +
                            " Props: " +
systemResponse[ii].getSignalDesc().getProperties().toString();
                        log.debug(s);
                    } else {
                        ++kk;
                    }
                }
                log.debug("kk=" + kk);
            }
```

```
            */
            // Get the infrastructure needs for this prediction interval
            try {
                if (_predictor.isReady() == false) {
                    if (arrivalRateStats.isAvailable()) {
                        double variance = arrivalRateStats.getVariance();
                        estimatedValues.setClusterValue(_webClusterId,
                                SignalType.ARRIVAL_RATE_VARIANCE.getName(),
                                variance);
                    }
                }

                needs[i] = _classifier.classifyInfrastructureNeeds(
                        signalValues,
                        estimatedValues,
                        _serviceLevelObjectives);
            } catch (ClassifierException ex) {
                throw new AppControllerException(ErrorCode.cannotEstimateInfrastructureNeeds,
_application.getName(), ex);
            }
        }
        // save it for MBean
        _infrastructureNeeds = needs;
        _signalValues = signalValues;
        if (mbserver != null) {
            mbeanHelper.registerMBeans (_infrastructureNeeds, _applicationId);
        }

        // check to see if we are in SLO breach
        checkSloBreachTrap (signalValues, _serviceLevelObjectives);
        return needs;
    } //-- estimateInfrastructureNeeds

    public long[] getPredictionIntervals() {
        return _predictionIntervals;
    }
    public void setPredictionIntervals(long[] newPredictionIntervals) {
        _predictionIntervals = newPredictionIntervals;
    }
    /**
     * Initialize according to the XML document
     * @param xmlConfig The XML document
     */
    public void initConfig(String xmlConfig) {
    }

    /**
     * Get the application name
     * @return The name
     */
    public String getName() {
        return _application.getName();
    }

    /**
     * @return The application associated with the AppController
     */
    public Application getApplication() {
        return _application;
    }

    // from Interface MBeanRegistration
    public ObjectName preRegister( MBeanServer server, ObjectName name )
        throws Exception
    {
        if (server != null) {
            mbserver = server;
            mbeanHelper.init(server);
        }
        else {
```

```
            log.warn("MBean Pre-registration cannot work with null server");
        }
        return name;
    }

    public void postRegister( Boolean registrationDone )
    {
        /** throws exception: WLM not compliant MBean
        mbeanHelper.registerMBean (_classifier, _applicationId);
        mbeanHelper.registerMBean (_wlm, _applicationId);
        */
        mbeanHelper.registerMBean (_predictor, _applicationId);
    }
    public void postDeregister()
    {
        mbeanHelper.unRegisterMBean (_predictor, _applicationId);
        mbeanHelper.unRegisterMBeans (_infrastructureNeeds, _applicationId);
    }
    /*
     * MBean implementations
     */
    public int getSamplingPeriod() {
        return samplingPeriod;
    }
    public void setSamplingPeriod(int value) {
        samplingPeriod = value;
    }

    public int getCalculationPeriod() {
        return calculationPeriod;
    }
    public void setCalculationPeriod(int value) {
        calculationPeriod = value;
    }
    public WorkLoadModel getAppWLM() {
        return _wlm;
    }
    public Predictor getPredictor() {
        return _predictor;
    }
    public AppClassifier getAppClassifier() {
        return _classifier;
    }
    public InfrastructureNeeds[] getInfrastructureNeeds() {
        return _infrastructureNeeds;
    }
    public void trainPrediction(Period history)
            throws AppControllerException {
        try {
            _predictor.trainPrediction(history, _predictionIntervals[_predictionIntervals.length
- 1]);
        } catch (PredictorException ex) {
            String[] messageParams = new String[2];
            messageParams[0] = _application.getName();
            messageParams[1] = ex.getMessage();
            throw new AppControllerException(ErrorCode.cannotTrainPrediction, messageParams, ex);
        }
    } //-- trainPrediction
    public Date getLastTrainPredictionTime() {
        return _predictor.getLastPredictionTrainingTime();
    } //-- getLastTrainPredictionTime
    public void onSample(TimeStampedValueSet valueSet) {
        if (valueSet == null) {
            return;
        }
        synchronized(_wlm) {
            //-- Train the predictor
            TimeStampedValue arrivalRate = valueSet.getClusterValue(_webClusterId,
                SignalType.ARRIVAL_RATE.getName());
            TimeStampedValue rawArrivalRate = valueSet.getClusterValue(_webClusterId,
```

```
                SignalType.RAW_ARRIVAL_RATE.getName());
            try {
                _predictor.trainOnline(arrivalRate);
            } catch (PredictorException ex) {
                log.error(ex.getLogString());
            }
            /**
             * Train the workload model. Even if training is disabled,
             * we need to call trainOnLine() at least once.
             */
            if ((_trainWLMInterval > 0) || (_trainingCount < 1)) {
                    if (System.currentTimeMillis() - _lastWLMTrainTime > _trainWLMInterval) {
                        _wlm.trainOnline(valueSet);
                        _lastWLMTrainTime = System.currentTimeMillis();
                        _trainingCount++;
                    }
            }

            if (_predictor.isReady() == false) {
                arrivalRateStats.newSample(rawArrivalRate.getValue());
            }
        }
/**      debugging to print out ArrivalRate signals.
        StringBuffer buf = new StringBuffer();
        buf.append ("clusterId="+_webClusterId);
        buf.append (", AR="+arrivalRate.valueToString());
        buf.append (", rawAR="+rawArrivalRate.valueToString());
        buf.append (", variance="+arrivalRateStats.varianceToString());
        log.info(buf.toString());
    }
    /**
     * for debugging only. Just to verify that the AppController instance
     * really gets garbage collected after being put in maintenance mode
     * and that there are no &quot;dangling referenced&quot; to these
     * objects.
     */
    public void finalize() {
        log.debug ("finalizing AppController for appId="+_application.getId());
    }
    /**
     * @return The SLA breach probability for the specified cluster.
     */
    public double getBreachProbabilityForCluster (int clusterId) {
        if ((_infrastructureNeeds == null) ||
        (_signalValues == null))
            return 0.0;
        InfrastructureNeeds currentNeeds = _infrastructureNeeds[0];
        if (currentNeeds == null)
            return 0.0;
        BreachProbabilityFunction bpf =
            currentNeeds.getSLABreachProbabilityFunction(clusterId);
        if (bpf == null)
            return 0.0;
        double numberOfServers = _signalValues.getClusterValue(
            clusterId, SignalType.NUMBER_OF_SERVERS).getValue();
        double breachProbability = bpf.classifyBreachProbability (
            (int) Math.round(numberOfServers));
        return breachProbability;
    }
   public double getNumberOfServersForCluster (int clusterId) {
        if (_signalValues == null)
            return 0.0;
        double numberOfServers = _signalValues.getClusterValue(
            clusterId, SignalType.NUMBER_OF_SERVERS).getValue();
        return numberOfServers;
    }
    public double getCpuUtilizationForCluster (int clusterId) {
        if (_signalValues == null)
            return 0.0;
        double cpuUtilization = _signalValues.getClusterValue(
```

```
            clusterId, SignalType.CPU_UTILIZATION).getValue();
        return cpuUtilization;
    }
    /**
     * Check to see if we are in SLO breach and if so call FaultManagement
     * to generate an SNMP trap.
     * <p>Internally refers to the _cpuSloTrapThreshold value initialized
     * by the AppControllerBean constructor. If the _cpuSloTrapThreshold
     * is set to -1, we use the criticalCPU as the threshold value.
     * @param signalValues used to fetch CPU utilization for each Cluster
     * @param slo used to fetch criticalCPU value for each Cluster
     */

    private void checkSloBreachTrap (TimeStampedValueSet signalValues,
        ServiceLevelObjectiveSet sloSet)
    {
        Iterator iter = _clusters.iterator();
        while (iter.hasNext()) {
            Cluster cluster = (Cluster) iter.next();
            int clusterId = cluster.getId();
            double cpuUtilization = _signalValues.getClusterValue(
                clusterId, SignalType.CPU_UTILIZATION).getValue();
            double trapThreshold = _cpuSloTrapThreshold;
            if (_cpuSloTrapThreshold < 0) { // use the sloSet criticalCPU
                trapThreshold =
                    sloSet.get(clusterId, SignalType.CRITICAL_CPU_UTILIZATION);
            }
            if (cpuUtilization >= trapThreshold) {
                FaultManagementFactory.getInstance().slaBreachEvent (_applicationId,
                    clusterId, "CPU-Utilization", trapThreshold, cpuUtilization);
            }
        }
    }
    /**
     * Fetches the CPU SLO trap threshold from the Properties table.
     * <p>First we try to find the threshold for the current Application
     * priority (with the name CPU-SLO-trap-threshold.priority=X).
     * If that does not exist we look for the default
     * CPU SLO trap threshold (CPU-SLO-trap-threshold).
     * If neither of these exist, we return -1, and the AppController
     * will use the criticalCPU value for the thresold.
     * @param applicationPriority application priority (1 to 10)
     * @returns CPU threshold value (0.0 to 1.0) or -1 if not available.
     */
    private final String SLO_THRESHOLD_BASENAME = "CPU-SLO-trap-threshold";
    private double fetchCpuSloTrapThreshold (int applicationPriority) {
        String keyName = SLO_THRESHOLD_BASENAME+".priority="+applicationPriority;
        double value = fetchPersistedDouble (DcmObjectId.KANAHA, keyName, -1);
        if (value == -1) {
            value = fetchPersistedDouble (DcmObjectId.KANAHA,
                SLO_THRESHOLD_BASENAME, -1);
        }
        return value;
    }
    /**
     * Fetches a value associated with the AppController
     * from the Properties table in the Database.
     * @param objectId ID of object that property is associated with.
     * @param key name of property
     * @param defaultValue value to return if property not found or invalid.
     */
     private double fetchPersistedDouble (DcmObjectId objectId, String key,
        double defaultValue)
     {
        String propertyStr = _dataCenterModel.getProperty (objectId,
            KanahaComponent.APPLICATION_CONTROLLER.getId(), key, null);
        double value = defaultValue;
        try {
            value = Double.parseDouble(propertyStr);
        } catch (Exception e) {
```

```
        }
        return (value);
    }
    /**
     * Persists an integer value associatied with the AppController
     * to the Properties table in the Database.
     * @param objectId ID of object that property is associated with.
     * @param key name of property
     * @param value value for property.
     */
    private void persist (DcmObjectId objectId, String key, double value)
        throws AppControllerException
    {
        try {
            _dataCenterModel.setProperty(objectId,
                KanahaComponent.APPLICATION_CONTROLLER.getId(),
                key, String.valueOf (value));
        }
        catch (DataCenterException ex) {
            throw new AppControllerException (ex.getErrorCode(), ex);
        }
    }
}
```

15.
```
public class QueueHybridProcessor extends AbstractProcessor {

/**
 * Description:  Hybrid processor which factors in the queueing model<br>
 */
   static Category log = Category.getInstance(QueueHybridProcessor.class.getName());
    // Output constants
    public static int UTILIZATION_RESULT = 0;
    public static int RESPONSE_TIME_RESULT = 1;
    public static int CRITICAL_UTILIZATION_RESULT = 2;
    public static int CRITICAL_RESPONSE_RESULT = 3;
    public static int CRITICAL_ARRIVAL_RESULT = 4;
    public static int NEXT_RESULT_INDEX = 5;
    // Configuration settings
    private int maxHistory = 50;
    private int minHistoryForTraining = 10;
    private double slaMaxCPU = .95; // In decimal
    private double slaMaxResponseTime = .167;
    private double outlierLimit = .05;
    // Index to outputs
    private int iUtilIdx = -1;
    private int iRespIdx = -1;
    private int iCritUtilIdx = -1;
    private int iCritRespIdx = -1;
    private int iCritArvlIdx = -1;
    private boolean bInitSignalIndex = false;
    // Outlier records
    private ArrayList outlierHistories[]; // Dimension of tier
    // Derivation parameters
    private double[][] serviceDemand; // Initial condition for modelling
    private double[][] cpuBase; // Initial condition for modelling
    private double[][] t_alpha_by_two; // Arbitary big value
    private double[][] sigmaValue; // Initial value very close to 0
    private double[][] Sxx; // Initial condition modelled from 3 samples
    private double[][] Syy; // Initial condition modelled from 3 samples
    private int[] numberServers; // Number of servers on each tier
    // Critical values
    private double criticalArrivalRate = 70; // Application critical arrival rate
    private double criticalCPU = .95;   // Bottleneck cluster critical CPU
    private double criticalResponseTime = .05; // Application critical response time
    private int criticalArrivalServers[]; // Number of server for critical arrival
    // Class to store history calculations
    private class HistoryCalculations {
        private double sumArrival;
        private double sumCpu;
```

```
        private double sumArrivalCpu;
        private double sumArrivalSquare;
        private double sumCpuSquare;
        private double slope;
        private double constant;
        public HistoryCalculations(double sumArrival, double sumCpu,
            double sumArrivalCpu, double sumArrivalSquare,
            double sumCpuSquare, double slope, double constant) {

            this.sumArrival     = sumArrival;
            this.sumCpu         = sumCpu;
            this.sumArrivalCpu  = sumArrivalCpu;
            this.sumArrivalSquare = sumArrivalSquare;
            this.sumCpuSquare   = sumCpuSquare;
            this.slope          = slope;
            this.constant       = constant;
        }

        public void setSumCpu(double sumCpu) {
            this.sumCpu = sumCpu;
        }
        public double getSumCpu() {
            return sumCpu;
        }
        public void setSumArrivalCpu(double sumArrivalCpu) {
            this.sumArrivalCpu = sumArrivalCpu;
        }
        public double getSumArrivalCpu() {
            return sumArrivalCpu;
        }
        public void setSumArrivalSquare(double sumArrivalSquare) {
            this.sumArrivalSquare = sumArrivalSquare;
        }
        public double getSumArrivalSquare() {
            return sumArrivalSquare;
        }
        public void setSumCpuSquare(double sumCpuSquare) {
            this.sumCpuSquare = sumCpuSquare;
        }
        public double getSumCpuSquare() {
            return sumCpuSquare;
        }
        public void setSumArrival(double sumArrival) {
            this.sumArrival = sumArrival;
        }
        public double getSumArrival() {
            return sumArrival;
        }
        public void setSlope(double slope) {
            this.slope = slope;
        }
        public double getSlope() {
            return slope;
        }
        public void setConstant(double constant) {
            this.constant = constant;
        }
        public double getConstant() {
            return constant;
        }
    }

    public QueueHybridProcessor() {
    }
    private void determineSignalIndex() {
        if (bInitSignalIndex == false) {
            // Lookup histrory index
            iUtilIdx = findSignalIndex(UTILIZATION_RESULT);
            iRespIdx = findSignalIndex(RESPONSE_TIME_RESULT);
            iCritUtilIdx = findSignalIndex(CRITICAL_UTILIZATION_RESULT);
```

```
            iCritRespIdx = findSignalIndex(CRITICAL_RESPONSE_RESULT);
            iCritArvlIdx = findSignalIndex(CRITICAL_ARRIVAL_RESULT);
            bInitSignalIndex = true;
        }
    }

    /**
     * Calculate the critical value for a particular tier class
     * @param iTier The index of the tier which critical value be calculated
     * @param iClass The work load class to be calculated
     */
    private void deriveCriticalValue(int iTier, int iClass) {
        // Prepare parameters to calculate critical rate (mule)
        double histCount = (double) histories[iTier].size();
        double avgArrival = getNormalizedAverageArrivalRate(iTier, iClass);
        double lastArrival = getNormalizedLatestArrivalRate(iTier, iClass);
        double rootTerm = Math.sqrt(1 + (1/histCount) +
            (Math.pow(lastArrival - avgArrival, 2) /
            Sxx[iTier][iClass]));
        log.debug("rootTerm: " + rootTerm);

        // Calculate critical arrival rate by solving quadratic equation
        // In the form of ax^2 + bx + c = 0 where:
        // a = roll*serviceDemand
        // b = roll*(1-cpuBase-fixTerm) - serviceDemand
        // c = - (cpuBase + fixTerm)
        double maxArrival = getNormalizedMaximumArrivalRate(iTier, iClass);
        double fixTerm = t_alpha_by_two[iTier][iClass] *
            sigmaValue[iTier][iClass] *
            Math.sqrt(1 + (1/histCount) +
            Math.pow(maxArrival - avgArrival, 2) /
            Sxx[iTier][iClass]);
        double valueA = - slaMaxResponseTime * serviceDemand[iTier][iClass]; // Always -ve
        double valueB = slaMaxResponseTime *
            (1 - cpuBase[iTier][iClass] - fixTerm) -
            serviceDemand[iTier][iClass];
        double valueC = - (cpuBase[iTier][iClass] + fixTerm);
        // Only cares about the bigger of the two solutions
        criticalArrivalRate = (- valueB - Math.sqrt(
            Math.pow(valueB, 2) - 4 * valueA * valueC)) /
            (2 * valueA);
        if (criticalArrivalRate <= 0) {
            log.error("Critical arrivate rate " + criticalArrivalRate +
                " is invalid.");
        }

        // Calculate critical CPU
        criticalCPU = cpuBase[iTier][iClass] +
            serviceDemand[iTier][iClass] * criticalArrivalRate;

        // Calculate critical Response Time for bottleneck tier
        criticalResponseTime =
            (criticalCPU + t_alpha_by_two[iTier][iClass] *
            sigmaValue[iTier][iClass] * fixTerm) /
            (criticalArrivalRate * (1 - criticalCPU
                - t_alpha_by_two[iTier][iClass] *
                sigmaValue[iTier][iClass] * fixTerm ));
        // Derive application's value by first scaling it back with the
        // current server count and then derive it back using output ratio.
        criticalArrivalServers = new int[totalTiers];
        for (int iTmpTier = 0; iTmpTier < totalTiers; iTmpTier++) {
            criticalArrivalServers[iTmpTier] = numberServers[iTmpTier];
        }
        criticalArrivalRate *= criticalArrivalServers[iTier] /
            getTierArrivalRatio(iTier);
        // Now this is the response time for one cluster, add the others...
        for (int iOtherTier = 0; iOtherTier < totalTiers; iOtherTier++) {
            if (iOtherTier != iTier) {
                // Derive pre-supposed arrival rate for that tier
                double otherArrivalRate = criticalArrivalRate *
```

```
                    getTierArrivalRatio(iOtherTier);
                // Normalize the arrival rate for calculations
                otherArrivalRate /= numberServers[iOtherTier];
                // Use 1/(mule - arrrate) to get other response time
                double otherResponseTime =
                    1 / (1/serviceDemand[iOtherTier][iClass]-otherArrivalRate);
                criticalResponseTime += otherResponseTime;
            } // else already calculated as bottleneck tier critical resp time
        }
        log.debug("criticalArrivalRate: " + criticalArrivalRate +
            ", criticalCPU: " + criticalCPU + " on tier index " +
            iTier + " and class " + iClass);
    }
    public int getBottleNeckTier() {
        int iBottleneckTier = 0;
        double worstDemand = 0;
        // Check the serviceDemand for the tier's class
        for (int iTier = 0; iTier < totalTiers; iTier++) {
            for (int iClass=0; iClass < totalWorkLoadClasses; iClass++) {
                double overallDemand = serviceDemand[iTier][iClass] *
                    getTierArrivalRatio(iTier);
                if (worstDemand < overallDemand) {
                    worstDemand = overallDemand;
                    iBottleneckTier = iTier;
                }
            }
        }
        return iBottleneckTier;
    }
    /**
     * Calculate the mean slope of the history distribution
     * @param histList  History lists which stores the history to be calculated
     * @param iTier     Tier index of history to be calculated
     * @param iClass    Class of servers
     * @param signType  Signal type for calculations
     * @param bFilterOutlier True to ignore any points which does not satisfy
     * current requirements.
     *
     * @return HistoryCalculations which holds calucated results.  May return
     * null if the calculation does not computer valid values.
     */
    private HistoryCalculations deriveSlopeAndConstant(ArrayList[] histList,
            int iTier, int iClass,
            SignalType signType, boolean bFilterOutlier) {
        /** @todo Replace this simple algorithm of comparing to mean */
        double variancePercentage = .1; // Allowed errors on slope difference
        double olSlope = 0;
        double olConstant = 0;
        double olSumArr = 0;
        double olSumCpu = 0;
        double olSumArrCpu = 0;
        double olSumArrSqr = 0;
        double olSumCpuSqr = 0;

        for (int iHistory=0; iHistory < histList[iTier].size(); iHistory++) {
            HistoryRecord olRecord =
                (HistoryRecord)histList[iTier].get(iHistory);
            double olArr = olRecord.getNormalizedArrivalRate(iClass);
            double olCpu = getHistoryDataValue(histList,
                iUtilIdx, iHistory, iTier, iClass,
                SignalType.CPU_UTILIZATION);
            // Avoid training in saturated area
            if (olCpu > criticalCPU) {
                olCpu = criticalCPU;
            }
            if (bFilterOutlier) {
                // Skip CPU if the value is a out-lier
                double theoreticalResult = serviceDemand[iTier][iClass] *
                    olArr + cpuBase[iTier][iClass];
                if (olCpu < .1) { // Bug 797 generates huge value at low CPU
```

```
                    histList[iTier].remove(olRecord); // Ignore points at small CPU
                    iHistory--; // Now every subsequent record shift back 1
                    continue;
                }
                else if (Math.abs(theoreticalResult - olCpu) > outlierLimit) {
                    // This is an out-lier point and no training
                    histList[iTier].remove(olRecord);
                    iHistory--; // Now every subsequent record shift back 1

                    log.info("Skipping out-lying point for tier " +
                        iTier + ", class " + iClass +
                        ". theoretical value " + theoreticalResult +
                        ", cpuValue " + olCpu);

                    if (outlierHistories[iTier].size() >=
                            minHistoryForTraining) {
                        outlierHistories[iTier].remove(0);
                    }
                    outlierHistories[iTier].add(olRecord);

                    continue; // Ignore this history, move on to next
                }
            }
            olSumArr += olArr;
            olSumCpu += olCpu;
            olSumArrCpu += (olArr * olCpu);
            olSumArrSqr += (olArr * olArr);
            olSumCpuSqr += (olCpu * olCpu);
        }

        double olSize = histList[iTier].size();
        if (olSize < minHistoryForTraining) {
            log.debug("History stored but not trained for tier " + iTier +
                " because total histories " + olSize +
                " does not meet mininmum requirement " +
                minHistoryForTraining);
            return null; // Do not return any value because it might well be invalid
        }
//        log.debug("Number of histories used for calculations on tier " +
//            iTier + " is " + olSize);

        if (olSumArr == 0 ||
            (olSize * olSumArrSqr - olSumArr * olSumArr) == 0) {
            return null; // Error with no arrival rate
        }
        else {
            olSlope = (olSize * olSumArrCpu - olSumArr * olSumCpu) /
                (olSize * olSumArrSqr - olSumArr * olSumArr);

            if (olSlope <= 0) {
                log.error("Training slope " + olSlope + " is invalid");
            }
        }
        olConstant = (olSumCpu - olSlope * olSumArr) / olSize;

        return new HistoryCalculations(olSumArr, olSumCpu, olSumArrCpu,
            olSumArrSqr, olSumCpuSqr, olSlope, olConstant);
    }

    public void updateModel() throws WorkloadModelException {
        // Using the history parameters and the SLA, the rest of parameters
        // can be derived
        determineSignalIndex();

        int iBottleneckTier = getBottleNeckTier();
        int iBottleneckClass = 0;

        double histCount = (double) histories[iBottleneckTier].size();
        if (histCount > 0) {
            HistoryRecord hrec = (HistoryRecord) histories[iBottleneckTier].get(
```

```
                (int) histCount-1);
            double maxTierArrival = getTierArrivalRatio(iBottleneckTier)
                * criticalArrivalRate;
            // Check for bottleneck tier ONLY to see if exceeding critical rate
            for (int iClass=0; iClass < totalWorkLoadClasses; iClass++) {
                // Note the number of servers may change and DO NOT use normalize values
                if ( hrec.getArrivalRate(iClass) / numberServers[iBottleneckTier] >
                        maxTierArrival / criticalArrivalServers[iBottleneckTier]) {
                    histories[iBottleneckTier].remove(hrec); // Invalid point
                    log.info("QueueHybridProcessor is unstable with tier " +
                        iBottleneckTier + ", class " + iClass + ". Server Arrival rate " +
                        hrec.getArrivalRate(iClass) / numberServers[iBottleneckTier] +
                        " greater than max arrival rate " +
                        maxTierArrival / criticalArrivalServers[iBottleneckTier]);
                    return;
                }
            }
        }

        // The first tier drives the arrival rate of subsequent tiers and
        // the first thing to determine is to locate the bottleneck of the
        // cluster chain by examining the System demand.
        for (int iTier = 0; iTier < totalTiers; iTier++) {
            histCount = (double) histories[iTier].size();
            if (histCount < minHistoryForTraining) {
                log.debug("History stored but not trained for tier " + iTier +
                    " because total histories " + histCount +
                    " does not meet mininmum requirement " +
                    minHistoryForTraining);
                continue; // Move to next tier, no training needed
            }

            histCount = (double) histories[iTier].size();
            for (int iClass=0; iClass < totalWorkLoadClasses; iClass++) {
                double sumArr = 0;      // (X = ArrivalRate)
                double sumCPU = 0;      // (Y = CPU Utilization)
                double sumArrCPU = 0;
                double sumArrSquared = 0;
                double sumCPUSquared = 0;
                double basicSlope = 0;
                double basicConstant = 0;

                HistoryCalculations basicHistories =
                    deriveSlopeAndConstant(histories, iTier, iClass,
                        SignalType.CPU_UTILIZATION, true);

                // Check whether the out-lier actually conforms
                // to a range which may be the real operating range
                boolean bUseOutliers = false;
                int outlierSize = outlierHistories[iTier].size();
                if (outlierSize >= minHistoryForTraining && // Minimum training number
                        histories[iTier].size() < outlierSize && // More out-liers than our
current points
                        histories[iTier].size() < maxHistory) { // Before stabilization
                    HistoryCalculations histCalc = deriveSlopeAndConstant(
                        outlierHistories, iTier, iClass,
                        SignalType.CPU_UTILIZATION, false);
                    if (histCalc != null) { // Make sure we have something valid here
                        double olMeanSlope = histCalc.getSlope();
                        double olConstant = histCalc.getConstant();

                        if (olMeanSlope > 0) {
                            // Check the percentage of points which is close to mean
                            Iterator olIter = outlierHistories[iTier].iterator();
                            int passCount = 0;
                            int failCount = 0;
                            while (olIter.hasNext()) {
                                HistoryRecord olRecord = (HistoryRecord) olIter.next();
                                double checkCpu =
                                    olRecord.getHistoryData(iClass, iUtilIdx);
```

```
                        double theoreticCpu =
                            olRecord.getNormalizedArrivalRate(iClass) *
                            olMeanSlope + olConstant;
                        if (Math.abs(checkCpu - theoreticCpu) >
                            outlierLimit) { // Check if it falls into limit range
                            failCount++;
                        }
                        else {
                            passCount++;
                        }
                    }

                    // Replace the current history is more points are closer to new mean
                    if ((double) passCount / (passCount + failCount) >= .6) {
                        log.info("A new operating range has been detected for tier " +
                            iTier + " with new service demand " +
                            histCalc.getSlope());

                        // Ready to jump to another operating range
                        sumArr = histCalc.getSumArrival();
                        sumCPU = histCalc.getSumCpu();
                        sumArrCPU = histCalc.getSumArrivalCpu();
                        sumArrSquared = histCalc.getSumArrivalSquare();
                        sumCPUSquared = histCalc.getSumCpuSquare();
                        basicSlope = histCalc.getSlope();
                        basicConstant = histCalc.getConstant();

                        // Switch to the outlier's history list...
                        histories[iTier] = outlierHistories[iTier];
                        outlierHistories[iTier] = new ArrayList();
                        bUseOutliers = true;
                    }
                    else { // This bunch of points are incorrect, throw half
                        log.debug("Veto failed for new opearting range " +
                            passCount + "/" + failCount +
                            ".  Discard half old points");
                        for (int iDiscard = 0;
                            iDiscard < minHistoryForTraining / 2;
                            iDiscard++) {
                                outlierHistories[iTier].remove(0);
                        }
                    }
                }
            }
        }

        if (bUseOutliers || basicHistories != null) {
            if (!bUseOutliers && basicHistories != null) {
                sumArr = basicHistories.getSumArrival();
                sumCPU = basicHistories.getSumCpu();
                sumArrCPU = basicHistories.getSumArrivalCpu();
                sumArrSquared = basicHistories.getSumArrivalSquare();
                sumCPUSquared = basicHistories.getSumCpuSquare();
                basicSlope = basicHistories.getSlope();
                basicConstant = basicHistories.getConstant();
            }

            // Derive subsequent values
            double sigmaSqaured = 0;
            sigmaValue[iTier][iClass] = 0;

            // Calculate the "best-fit" serviceDemand and cpuBase
            serviceDemand[iTier][iClass] = basicSlope;
            log.debug("Service Demand for tier " + iTier +
                ", class " + iClass + ": " +
                serviceDemand[iTier][iClass]);
            cpuBase[iTier][iClass] = basicConstant;

            // Calculate the model error variance (Sigma square)
            Sxx[iTier][iClass] = sumArrSquared - (sumArr*sumArr / histCount);
```

```
                    Syy[iTier][iClass] = sumCPUSquared -
                        (sumCPU*sumCPU / histCount);
                    double Sxy = sumArrCPU - (sumArr*sumCPU / histCount);
                    sigmaSqaured = (Syy[iTier][iClass] -
                        (serviceDemand[iTier][iClass] * Sxy)) / (histCount - 2);
                    if (sigmaSqaured < 0) {
                        String msg = "Negative sigma square value: " +
                            sigmaSqaured;
                        if (sigmaSqaured < -.01) {
                            WorkloadModelException wlme = new
                                WorkloadModelException(
                                ErrorCode.wlmInvalidInputValues, msg);
                            log.warn(wlme.getLogString());
                        }
                        else
                            log.warn(msg + " probably due to precision error.");
                        sigmaSqaured = 0;
                    }
                    sigmaValue[iTier][iClass] = Math.sqrt(sigmaSqaured);

                    // Calculate the alpha value from SLA
                    double alpha = (1 + slaMaxCPU) / 2;
                    t_alpha_by_two[iTier][iClass] = 0;

                    // Check for history size to perform appropriate operation
                    if (histCount <= 32) { // Perform T-Inverse function
                        t_alpha_by_two[iTier][iClass] = getTInverse(alpha,
                            histCount - 2);
                    }
                    else { // Normal distribution
                        t_alpha_by_two[iTier][iClass] = getNormalInverse(alpha);
                    }
                }
            }
        }

        iBottleneckTier = getBottleNeckTier();
        for (int iTier = 0; iTier < totalTiers; iTier++) {
            if (histories[iTier].size() < 1) {
                // All histories got filtered by out-liers
                log.warn("Out-lier filtering results in tier " +
                    + iTier + " with no history.");
                return;
            }
        }

        // Handle calculations of critical values with bottleneck tier
        deriveCriticalValue(iBottleneckTier, iBottleneckClass);
    }

    public void subscribeOutput(int outputIndex, SignalType trainingSignal)
            throws WorkloadModelException {

        super.subscribeOutputCheckIndex(outputIndex, trainingSignal,
            NEXT_RESULT_INDEX);
    }

    public ProcessorInfo[] storeProcessorParameters() {
        // Setup cluster_id in hashmap
        ProcessorInfo[] piTiers = super.storeProcessorParameters();

        // For each tier stores the existing parameters...
        for (int iTier = 0; iTier < totalTiers; iTier++) {
            if (histories[iTier].size() >= minHistoryForTraining) {
                for (int iClass = 0; iClass < totalWorkLoadClasses; iClass++) {
                    Properties tierProps = (Properties)
                        piTiers[iTier].getProperties();
                    tierProps.setProperty("qh_cpuServiceDemand",
                        String.valueOf(serviceDemand[iTier][iClass]));
                    tierProps.setProperty("qh_cpuBase",
```

```
                    String.valueOf(cpuBase[iTier][iClass]));
                tierProps.setProperty("qh_tAlphaByTwo",
                    String.valueOf(t_alpha_by_two[iTier][iClass]));
                tierProps.setProperty("qh_sigma",
                    String.valueOf(sigmaValue[iTier][iClass]));
                tierProps.setProperty("qh_sxx",
                    String.valueOf(Sxx[iTier][iClass]));
                tierProps.setProperty("qh_syy",
                    String.valueOf(Syy[iTier][iClass]));
            }
        }
    }

    return piTiers;
}

public void configureProcessor(int iProcessorId, int totalServiceClass,
        HashMap configProps[]) {
    // Generic processing
    super.configureProcessor(iProcessorId, totalServiceClass, configProps,
        maxHistory);

    // Setup core signals
    coreSignalList.add(SignalType.CPU_UTILIZATION);

    // Retrieve sla parameters from first cluster
    slaMaxCPU = getClusterProperty(configProps[0],
        0, "slaMaxCPU", .95);
    slaMaxResponseTime = getClusterProperty(configProps[0],
        0, "slaMaxResponseTime", .167);
    outlierLimit = getClusterProperty(configProps[0],
        0, "outlier-limit", .05);

    // Setup initial values of needed parameters
    serviceDemand = new double[totalTiers][totalServiceClass];
    cpuBase = new double[totalTiers][totalServiceClass];
    t_alpha_by_two = new double[totalTiers][totalServiceClass];
    sigmaValue = new double[totalTiers][totalServiceClass];
    Sxx = new double[totalTiers][totalServiceClass];
    Syy = new double[totalTiers][totalServiceClass];
    outlierHistories = new ArrayList[totalTiers];

    numberServers = new int[totalTiers];
    criticalArrivalServers = new int[totalTiers];

    // Initial condition for modelling
    for (int iTier = 0; iTier < totalTiers; iTier++) {
        numberServers[iTier] = 1;
        criticalArrivalServers[iTier] = 1;
        outlierHistories[iTier] = new ArrayList();
        for (int iClass = 0; iClass < totalServiceClass; iClass++) {
            serviceDemand[iTier][iClass] = getClusterProperty(
                configProps[iTier], iClass,
                "qh_cpuServiceDemand", .0129);// .0258);
            cpuBase[iTier][iClass] = getClusterProperty(configProps[iTier],
                iClass, "qh_cpuBase", .0037);
            t_alpha_by_two[iTier][iClass] = getClusterProperty(
                configProps[iTier], iClass, "qh_tAlphaByTwo", 30);
            sigmaValue[iTier][iClass] = getClusterProperty(
                configProps[iTier], iClass, "qh_sigma", 0);
            Sxx[iTier][iClass] = getClusterProperty(
                configProps[iTier], iClass, "qh_sxx", 200);
            Syy[iTier][iClass] = getClusterProperty(
                configProps[iTier], iClass, "qh_syy", .033282);
        }
    }
}

public double[][][] deriveOutput(double[] arrivalRates)
        throws WorkloadModelException {
```

```
// Generic checking
double[][][] output = super.deriveOutput(arrivalRates);

// Check history size
for (int iTier = 0; iTier < totalTiers; iTier++) {
    if (histories[iTier].size() == 0) {
        throw new WorkloadModelException(ErrorCode.wlmInsufficientHistory,
            "There is no valid history to generate data on tier " +
            iTier);
    }
}

// Pre-determined subscription index...
int iTotalOut = 0;

Iterator subscribeIter = subscribedList.iterator();
determineSignalIndex();

for (int iTier = 0; iTier < totalTiers; iTier++) {
    for (int iClass=0; iClass < totalWorkLoadClasses; iClass++) {
        double avgArrival = getNormalizedAverageArrivalRate(iTier, iClass);
        double lastArrival = arrivalRates[iClass] / numberServers[iTier]
            * getTierArrivalRatio(iTier);

        // Calculate the CPU%
        if (iUtilIdx != -1) {
            // Check for critical APPLICATION arrival rate
            if (criticalArrivalRate * getTierArrivalRatio(iTier)
                    > lastArrival) {
                // Return cluster's CPU of interest
                output[iTier][iClass][iUtilIdx] =
                    cpuBase[iTier][iClass] +
                    serviceDemand[iTier][iClass] * lastArrival;
            }
            else { // If exceeding critical rate, clip it
                output[iTier][iClass][iUtilIdx] = slaMaxCPU;
            }
        }

        // Response time calculation
        if (iRespIdx != -1) {
            if (serviceDemand[iTier][iClass] != 0) {
                // Calculate service rate (mule)
                double rootTerm = Math.sqrt(1 +  1 /
                    histories[iTier].size() +
                    Math.pow(lastArrival - avgArrival, 2) /
                    Sxx[iTier][iClass]);

                double commonTerm = serviceDemand[iTier][iClass] * lastArrival +
                    t_alpha_by_two[iTier][iClass] *
                    sigmaValue[iTier][iClass] * rootTerm;
                double mule = lastArrival /
                    (cpuBase[iTier][iClass] + commonTerm) ;
                output[iTier][iClass][iRespIdx] = 1 /
                    (mule - lastArrival);
            }
            else { // Use last historical value if not tuned properly
                HistoryRecord hr = (HistoryRecord) histories[iTier].get(
                    histories[iTier].size() - 1);
                output[iTier][iClass][iRespIdx] =
                    hr.getHistoryData(iClass, iRespIdx);
            }
        }

        // Arrival Rate calculations independent of new arrival rate
        if (iCritArvlIdx != -1) {
            output[iTier][iClass][iCritArvlIdx] =
                criticalArrivalRate * getTierArrivalRatio(iTier);
        }
```

```
                // CriticalCPU return the application value
                // independent of clusters
                if (iCritUtilIdx != -1) {
                    output[iTier][iClass][iCritUtilIdx] = criticalCPU;
                }

                // Critical Response Time returns the application value
                // independent of clusters (i.e. Overall response time)
                if (iCritRespIdx != -1) {
                    // Response time = 1/(mule-arrrate)
                    output[iTier][iClass][iCritRespIdx] = criticalResponseTime;
                }
            }
        }
        return output;
    }

   /**
     * Return the inverse T-distribution of probability
     *
     * @param proabability The probability of the T-distribution
     * @param degereeFeedom Degree of freedom in the T-distribution
     * @return The T-value for generating the the probability given degree of freedom
     */
    private double getTInverse(double probability, double degreeFreedom) {
        // Use a temporary lookup table for now...
        double calcValue = StatFunctions.qt(probability, degreeFreedom, true);
        log.debug("T-inverse of " + probability + " with degree of freedom "
            + degreeFreedom + " = " + calcValue);
        return calcValue;
    }

    /**
     * Return the inverse normal distribution of probability
     *
     * @param proabability The probability of the normal distribution
     * @retrun The Normal-value for generating the probability
     */
    private double getNormalInverse(double probability) {
        double calcValue = StatFunctions.qnorm(probability, false);
        log.debug("Normal-inverse of " + probability + " = " + calcValue);
        return calcValue;
    }

    public void updateModelServers(int[] servers)
            throws WorkloadModelException {
        super.updateModelServers(servers);

        // The number of servers change the arrival rate modelling here...
        numberServers = servers;
    }

    public String getStrId() {
        throw new java.lang.UnsupportedOperationException("Method getStrId() not yet
implemented.");
    }
}
```

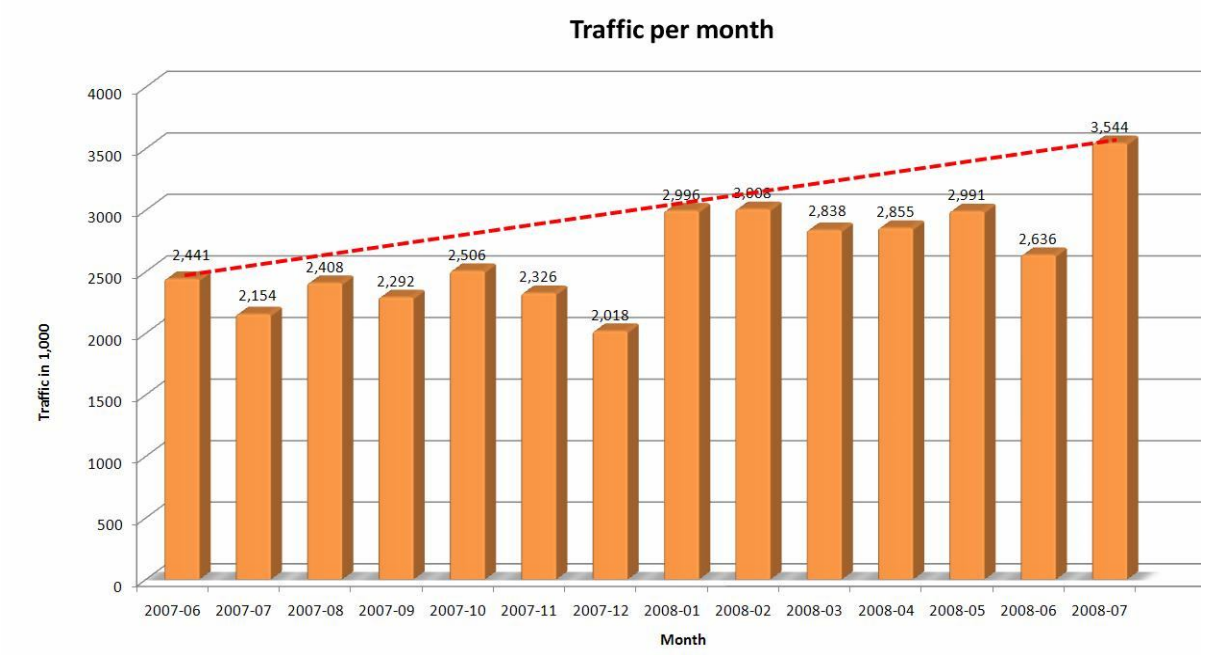
ANEXA 2 SIMULAREA COMPORTAMENTULUI DINAMIC AL UNUI SISTEM AUTOADMINISTRAT

În Anexa 2 se prezintă informația care descrie comportarea dinamică a unui sistem complex de calculatoare, tipic pentru un centru de calcul modern. Acest tip de aplicații şi problemele generate de necesitatea funcționării sale optimale cu un cost redus a reprezentat motivația principală pentru eforturile de cercetare şi rezultatele descrise în această teză.

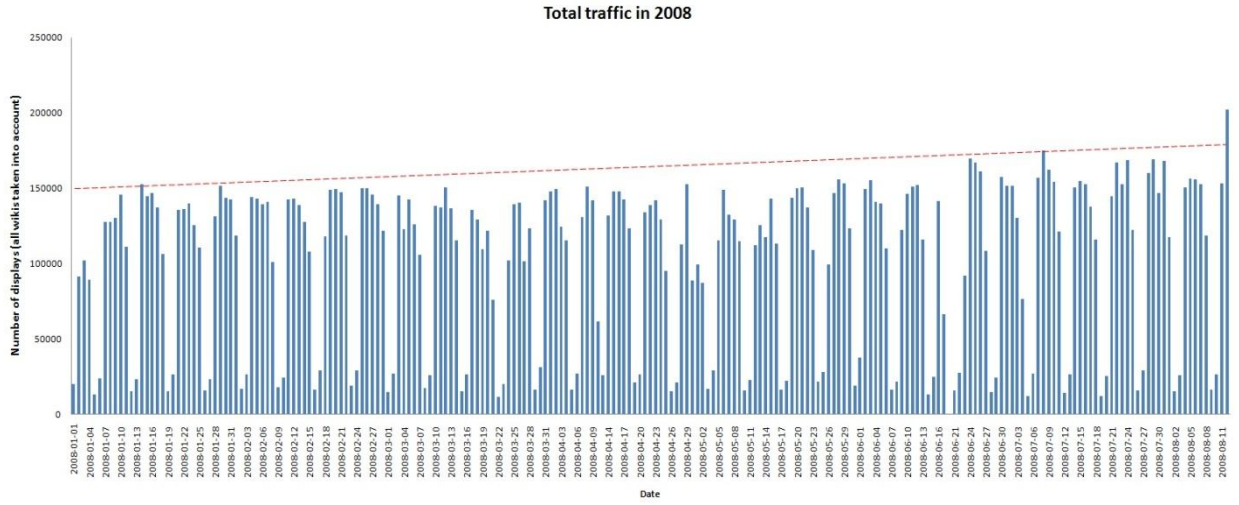
Sistemul prezentat este IBM Wiki Central, cuprinzând peste 25000 de wiki şi rulând pe 9 servere. Datele au fost colectate între iunie 2007 (două servere, alocare statică) şi august 2008 (9 servere, alocare dinamică) şi au fost extrase prin procesarea a peste 200 GB de loguri.

Rezultatele aplicării unui sistem de ajustare automată a numărului de servere alocate acestei aplicații sunt evidente în graficele de utilizare prezentate.

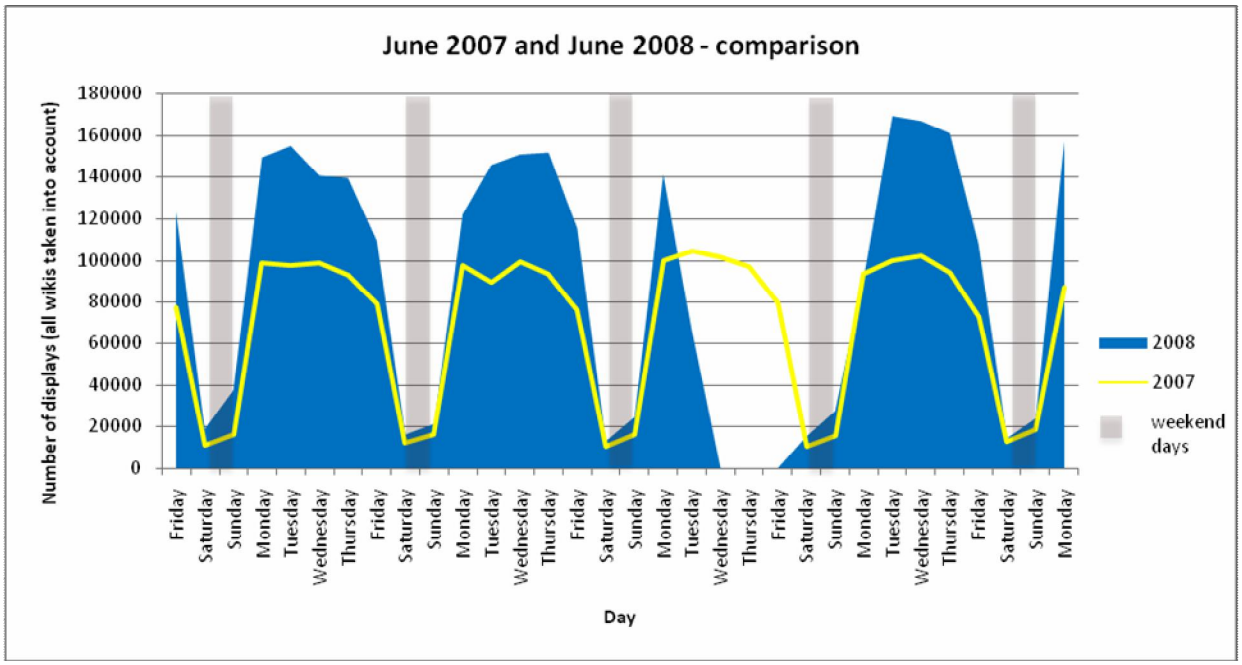
1. Histograma traficului de rețea lunar 2007 - 2008 pentru toți utilizatorii



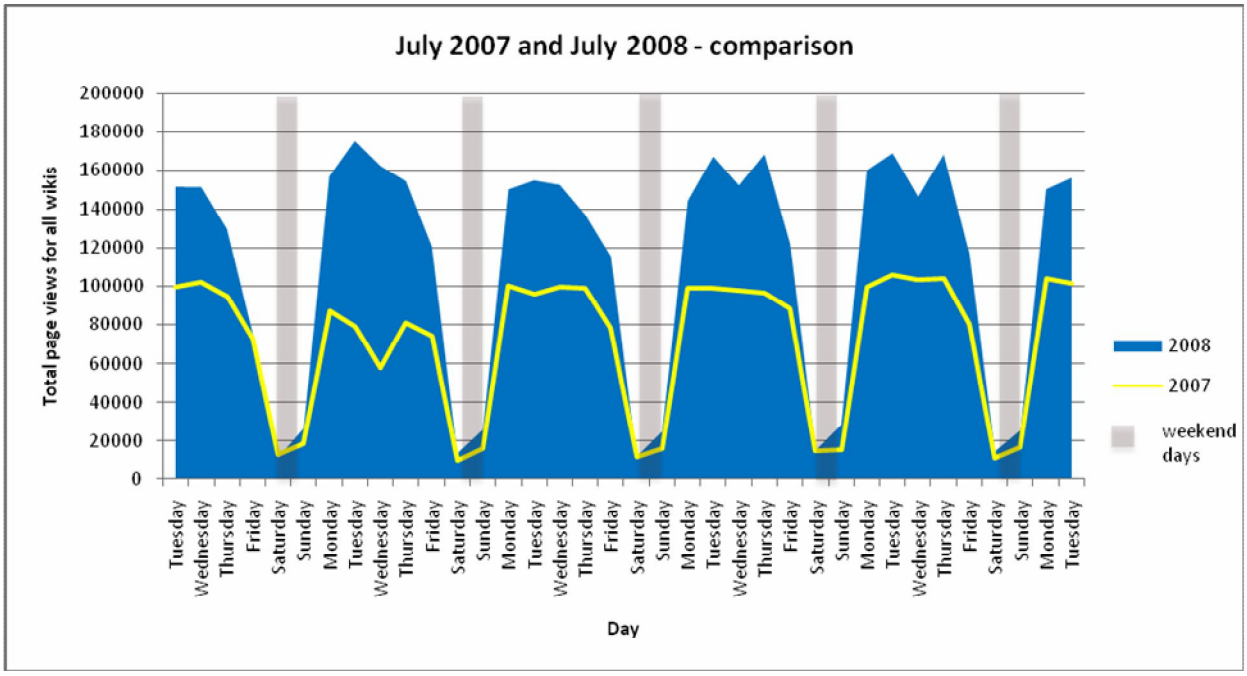
2. Histograma traficului de rețea zilnic în 2008 pentru toți utilizatorii

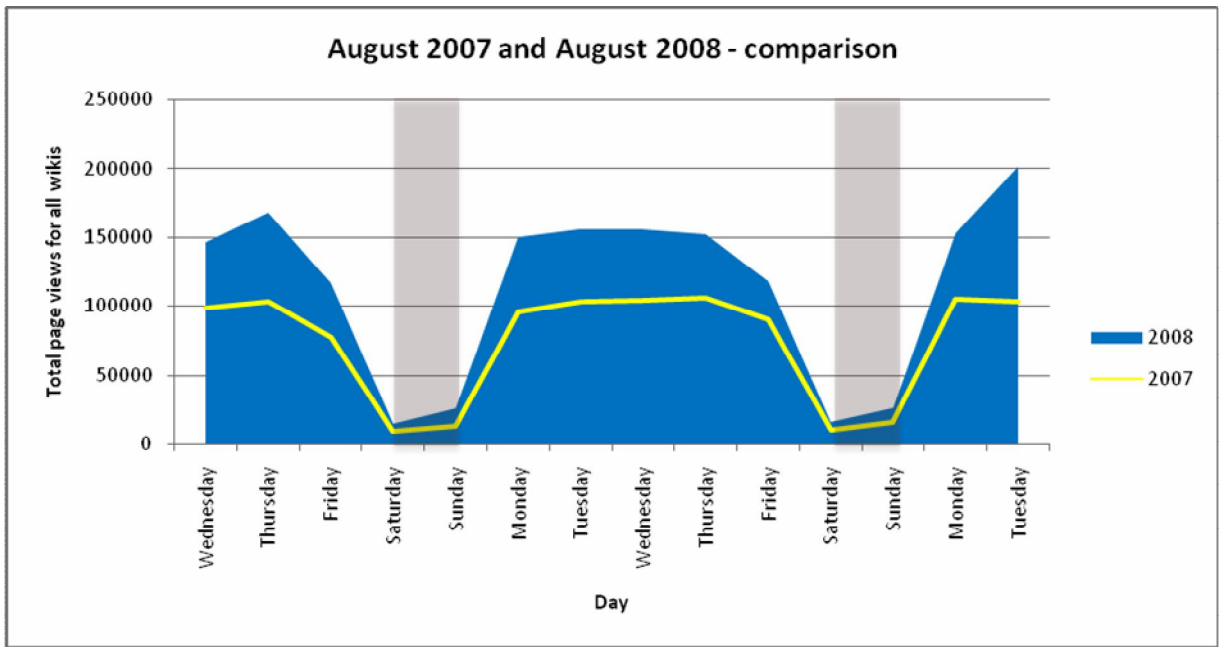


3. Comparație între traficul de rețea în iunie 2007 şi iunie 2008. Prin aplicarea sistemului de alocare dinamică, traficul de rețea aproape s-a dublat.

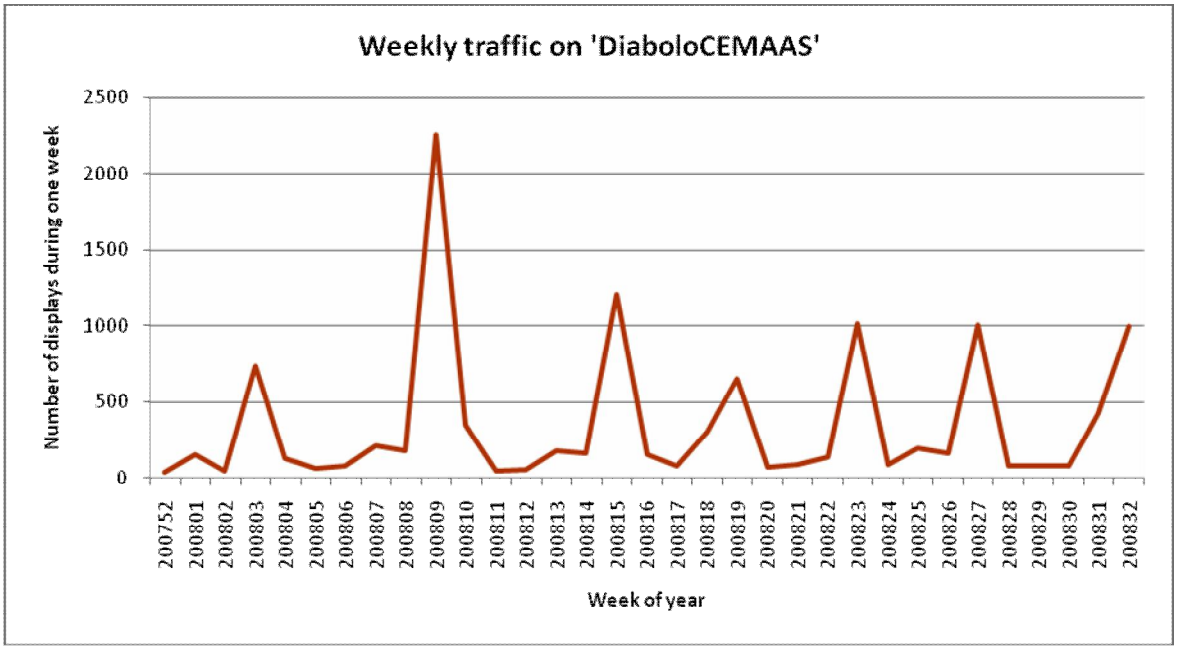


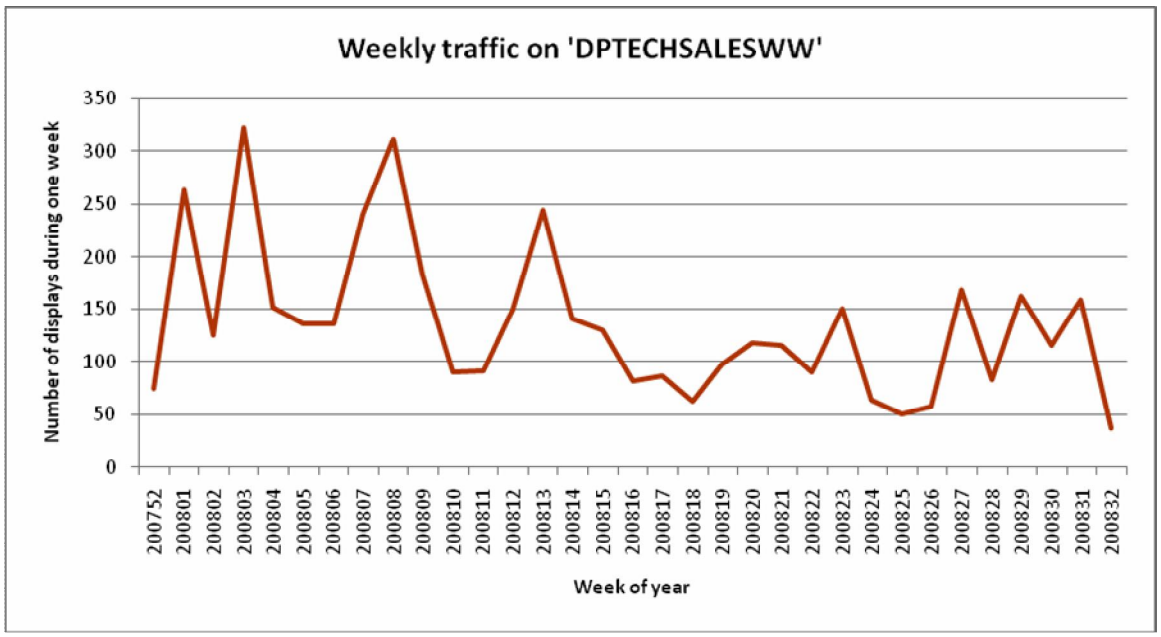
4. Similar cu graficul precedent, comparație între iulie/august 2007 şi iulie/august 2008. Rezultatele sunt similare, demonstrând că sistemul este stabil în condiții diferite de trafic.

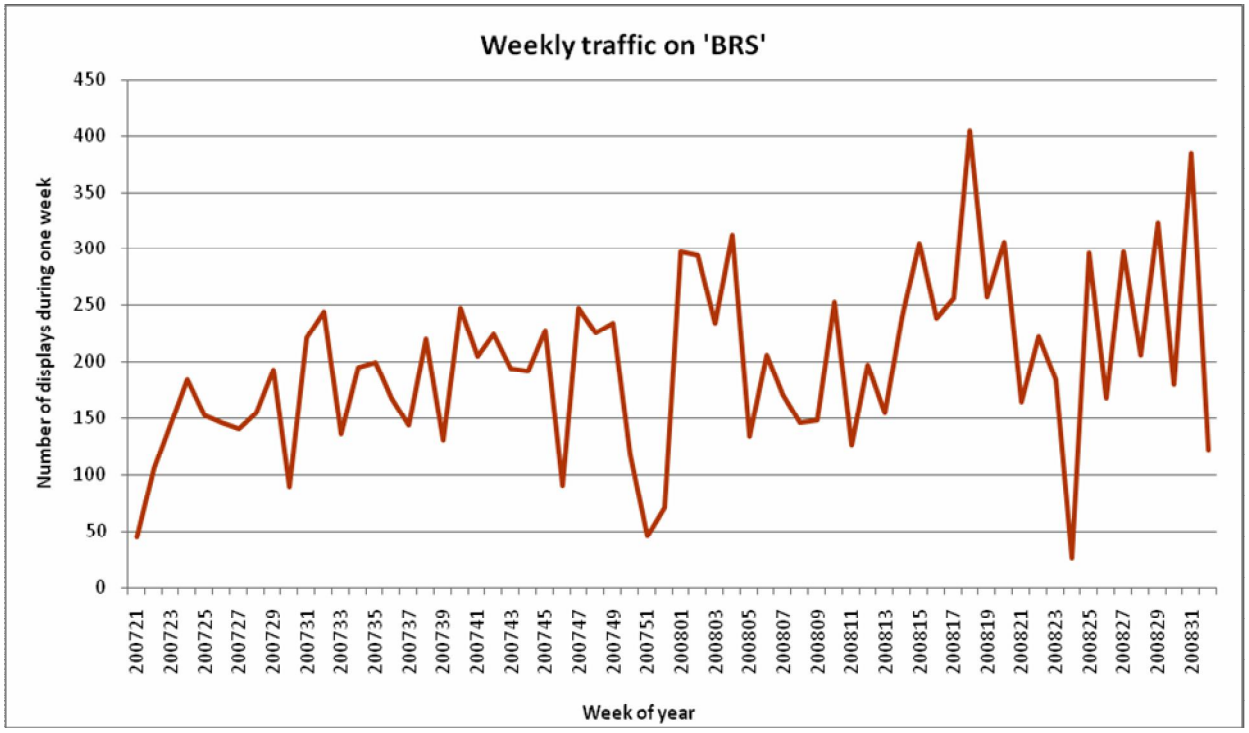


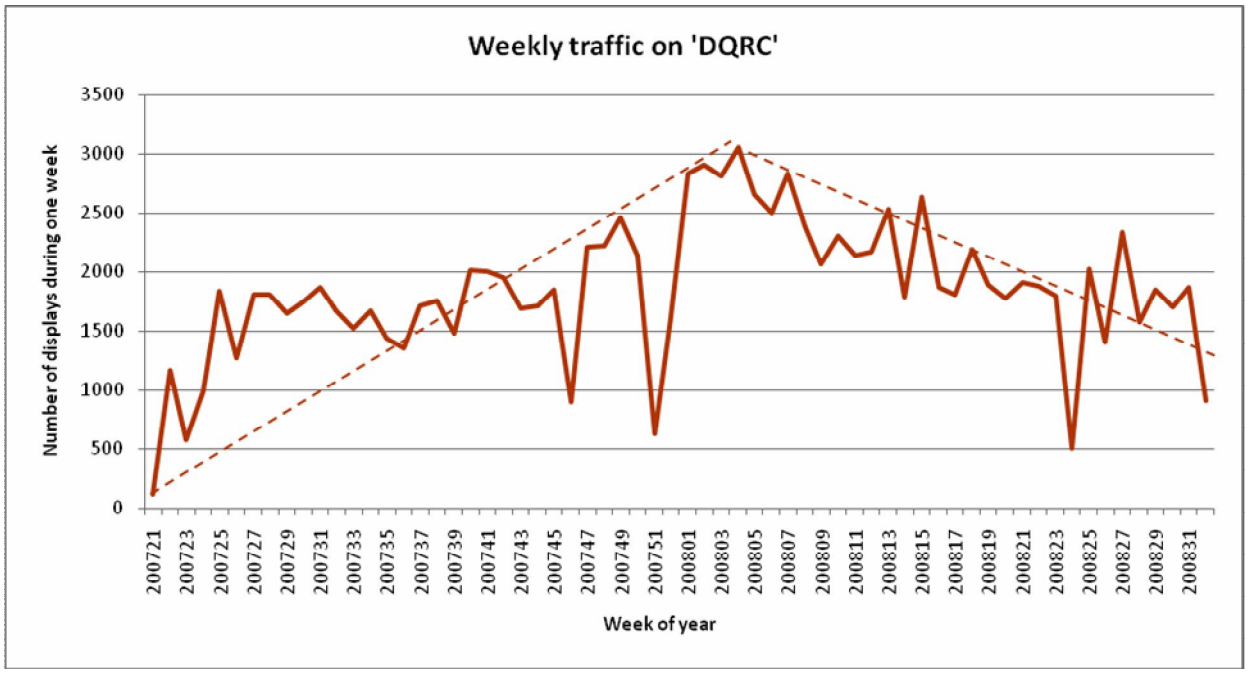


5. Trafic de rețea pentru 4 wiki cu caracteristici diferite, ilustrând caracterul complex al sistemului şi necesitatea aplicării metodelor descrise în teză.

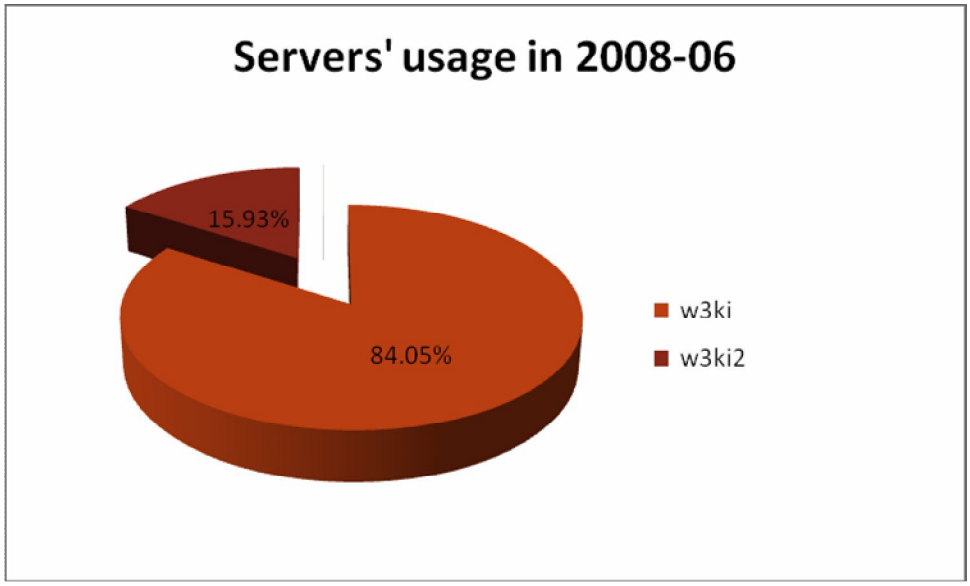

Weekly traffic on 'DiaboloCEMAAS'
Number of displays during one week
2500
2000
1500
1000
500
0
200752
200801
200802
200803
200804
200805
200806
200807
200808
200809
200810
200811
200812
200813
200814
200815
200816
200817
200818
200819
200820
200821
200822
200823
200824
200825
200826
200827
200828
200829
200830
200831
200832
Week of year

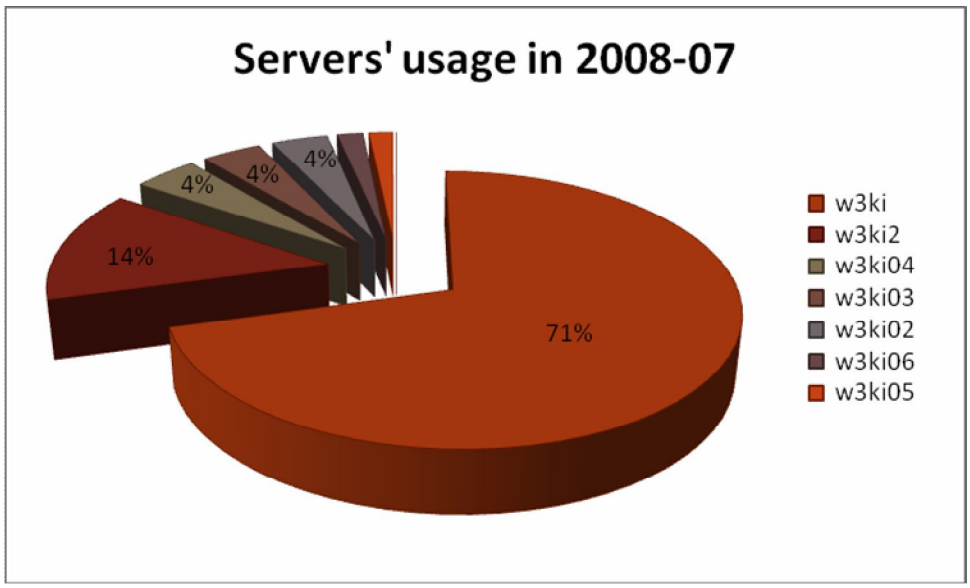

Weekly traffic on 'DPTECHSALESWW'
Number of displays during one week
350
300
250
200
150
100
50
0
200752
200801
200802
200803
200804
200805
200806
200807
200808
200809
200810
200811
200812
200813
200814
200815
200816
200817
200818
200819
200820
200821
200822
200823
200824
200825
200826
200827
200828
200829
200830
200831
200832
Week of year





6. Situația utilizării servelor în iunie şi iulie 2008. La începutul lui iulie 2008, Wiki Central a fost modificat să includă un sistem de alocare dinamică a servelor similar cu cel descris în teză.


Servers' usage in 2008-06
15.93%
84.05%
w3ki
w3ki2


Servers' usage in 2008-07
14%
4%
4%
4%
71%
w3ki
w3ki2
w3ki04
w3ki03
w3ki02
w3ki06
w3ki05

BIBLIOGRAFIE

Din multitudinea lucrărilor publicate în domeniul Disponibilității Sistemelor Digitale Complexe bazate pe reconfigurarea autonomă dinamică, am citat în această teză de doctorat numai monografiile, cărțile tehnice şi lucrările ştiințifice publicate după anul 2000.

A. Tratate, Monografii, Reviste, Proceedinguri.

A.1. [AB-04a] A. Brown, J. Hellerstein, M. Hogstrom, T. Lau, S. Lightstone, P. Shum, and M. Yost. "Benchmarking autonomic capabilities: Promises and pitfalls." In *Proceedings of the* First International Conference on Autonomic Computing, IEEE Computer Society, 2004, pp. 266–267.

A.2. [AB-04b] A. Beygelzimer, G. Grinstein, R. Linkser, and I. Rish." Improving network robustness." In Proceedings of the First International Conference on Autonomic Computing, pp. 322–323. IEEE Computer Society, 2004.

A.3. [AB-05a] A.A. Bougaev, " Pattern recognition based tools enabling autonomic computing." In Proceedings of 2nd IEEE International Conference on Autonomic Computing (ICAC), ICAC 2005, pp.313–314

A.4. [AB-05b] A. Brown and C. Redlin. "Measuring the effectiveness of self-healing autonomic systems." In Proceedings of the Second International 21 Conference on Autonomic Computing. IEEE Computer Society, 2005, pp. 1412-1427.

A.5. [AC-05] A. Casavola, M. Giannelli, and E. Mosca, "Min-max predictive control strategies for input saturated polytopic uncertain systems," *Automatica*, vol. 54, no. 7, 2005, pp. 627 – 643.

A.6. [AD-04] A. Dan *et al.*, "Web Services on Demand: WSLA-Driven Automated Management," *IBM Systems Journal*, vol. 43, no. 1, 2004, pp. 136–158.

A.7. [AF-00] A. Finkelstein and J. Kramer, "Software engineering: a roadmap", Proceedings of the Conference on The Future of Software Engineering, ACM Press, Limerick, Ireland, 2000, pp. 158-172.

A.8. [AF-03] A. Fox, D. Patterson: "Self-repairing computers", Ed. Scientific American, N.Y. 2003

A.9. [AG-06] A. Ganek, R.J.Friedrich, "The road ahead—achieving wide-scale deployment of autonomic technologies." In Proceedings of the 3rd IEEE International Conference on Autonomic *Computing*. Dublin, Ireland, 2006, pp.799-818

A.10. [AK-03] A. Keller and H. Ludwig, "The WLSA Framework: Specifying and Monitoring Service Level Agreements for Web Services", Journal of Network and Systems Management, Special Issue on EBusiness Management, 11:1, Plenum Publishing Corp, USA 2003,pp.423-430.

A.11. [AK-04] A. Keller, J. Hellerstein, J. Wolf, K. Wu, and V. Krishnan. „The CHAMPS system: Change management with planning and scheduling." In *Proceedings of the IEEE/ IFIP Network Operations and Management Symposium*. Kluwer Academic Publishers, 2004, pp.1015-1034.

A.12. [AL-01] A. van Lamsweerde, "Goal-Oriented Requirements Engineering: A Guided Tour", Proceedings of the 5th IEEE International Symposium on Requirements Engineering, IEEE Computer Society, USA, 2001,pp.453-468.

A.13. [AM-05] J.A.McCann, A.Navarra and A. Papadopoulos, "Connectionless probabilistic (CoP) routing: An efficient protocol for mobile wireless ad-hoc sensor networks", In *Proceedings of 24th IEEE International Performance, Computing and Communications Conference*. Phoenix, AZ, USA 2005, pp.779-792.

A.14. [AM-06] J.A. McCann, M. Huebscher, "Context as autonomic intelligence in a ubiquitous computing environment". *Int. J. Intern. Protocol Tech.* Special Edition on Autonomic Computing. USA 2006, pp.918-943.

A.15. [AP-02] A. Ponnappan, L.Yang, R. Pillai and P. Braun, "A policy based QoS management system for the intserv/diffserv based internet", In Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks. USA 2002,pp.159–168.

A.16. [AT-02] A. Tannenbam, "*Computer Networks", Ed.* Prentice Hall, NY, USA. 4th ed.,2002.

A.17. [AV-06] A. McVeigh, J. Kramer and J. Magee, "Using resemblance to support component reuse and evolution" , Proceedings of the 2006 conference on Specification and verification of component-based systems, ACM Press, Portland, Oregon, USA 2006,pp.318-332.

A.18. [AW-00a] A.Wise, A.G. Cass, B.S. Lerner, E.K. Call, L.J.Osterweil, "Using Little-JIL to coordinate agents in software engineering". In Proceedings of the Automated Software Engineering Conference (ASE'00), USA 2000, pp.78-95.

A.19. [AW-00b] A. Wolf, D.Heimbigner, J. Knight, P.Devanbu, M.Gertz and A.Carzaniga,"Bend, don't break: Using reconfiguration to achieve survivability", In *Proceedings of the 3rd Information Survivability Workshop*, Seattle, USA 2000, pp. 338-349.

A.20. [BC-02] J.M. Cobleigh, L.J.Osterweil, A. Wise and B. Lerner, "Containment units: A hierarchically composable architecture for adaptive systems", *SIGSOFT Softw. Engin. Notes 27*, 6, USA 2002, pp.159–165.

A.21. [BD-02] B.P. Douglass, "Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems", Addison Wesley Professional, USA 2002.

A.22. [BJ-01] D.B. Johnson, D.A. Maltz and J. Broch, " DSR: The dynamic source routing protocol for multihop wireless ad hoc networks". In *Ad Hoc Networking*, C. Perkins, Ed. Addison-Wesley, Chapter 5, 2001, pp.139–172.

A.23. [BK-06] B. Khargharia, S. Hariri and M. Yousif," Autonomic power and performance management for computing systems". In Proceedings of 3rd IEEE International Conference on Autonomic Computing (ICAC). Dublin, Ireland 2006, pp. 145–154.

A.24. [BM-00] B. Moore, J. Strassner, and E. Eileson, "Policy Core Information Model Version 1 Specification", Technical report, IETF, USA 2000.

A.25. [BM-04] B.Melcher and B.Mitchell," Towards an autonomic framework: Self-configuring network services and developing autonomic applications", *Intel Techn. J. 8*, 4,USA 2004,pp. 279–290.

A.26. [BS-05] B. Simmons and H. Lutfiyya, "Policies, Grids and Autonomic Computing. Design and Evolution of Autonomic Application Software" , Workshop associated with the 27th International Conference on Software Engineering (ICSE), DEAS 2005.

A.27. [BS-06] B. Simmons, H. Lutfiyya, M. Avram and P. Chen "A Policy-Based Framework for Managing Data Centers", In Proceedings of 2006 IEEE/IFIP Network Operations & Management Symposium (NOMS 2006), Vancouver, Canada 2006, pp.412-432.

A.28. [BS-06a] B. Solomon, D. Ionescu, M. Litoiu, M. Mihaescu. "Decentralized Predictive Control of Autonomic Computing Environments", 4th International Information and Telecommunication Technologies Symposium (I2TS'2006),Cuiaba, Brazil 2006, pp. 94-103.

A.29. [BS-07] B. Solomon, D. Ionescu, M. Litoiu, M. Mihaescu. "Towards a Real-Time Reference Architecture for Autonomic Systems." SEAMS Workshop, 29th International Conference on Software Engineering and Workshops (ICSE-ICSE Workshops), 2007, pp. 1224-1231.

A.30. [BS-07a] B.Solomon, D.Ionescu, M.Litoiu, M.Mihaescu: "A Real-Time Pattern Based Approach to Autonomic Computing" SEAMS 2007 ACM Workshop on Software Engineering for Adaptive and Self-Managing Systems, Minneapolis, Minnesota, USA 2007,pp.78-85.

A.31. [BS-07b] B.Solomon, D.Ionescu, M.Litoiu, M.Mihaescu:" Adaptive Autonomic Computing" Intern. Symp.CACON, Toronto, CA.2007, pp.301-306.

A.32. [BS-07c] B. Solomon, D. Ionescu, M. Litoiu, and M. Mihaescu, "An autonomic computing approach to server virtualization", 5th International Information and Telecommunication Technologies Symposium, Ontario 2007, pp. 87-94.

A.33. [BS-07d] B. Solomon, D. Ionescu, M.Litoiu and M. Mihaescu, "A real-time adaptive control of autonomic computing environments". In Proceedings of the 2007 Conference

of the Center For Advanced Studies on Collaborative Research (Richmond Hill, Ontario, Canada, 2007). NY,USA 2007, pp. 124-136.
DOI=http://doi.acm.org/10.1145/1321211.1321225

A.34. [BU-05] B.Urgaonkar, P. Shenoy, A.Chandra and P.Goyal, "Dynamic provisioning of multi-tier Internet applications", In Proceedings of the 2nd International Conference on Autonomic Computing (ICAC). ICAC 2005, pp.217–228.

A.35. [CA-02] C. Amza, A. Ch, A. Cox, S. Elnikety, R. Gil, K. Rajamani, E. Cecchet, and J. Marguerite. "Specification and implementation of dynamic Web site benchmarks", In *Proceedings of WWC-5: IEEE 5th Annual Workshop on Workload Characterization*, LA,USA 2002, pp.178-198.

A.36. [CA-05] C. Amza , G. Soundararajan , "Online Data Migration for Autonomic Provisioning of Databases in Dynamic Content Web Servers," *Proceedings of CASCON 2005, ACM Digital Library,* Ontario, Canada 2005, pp. 261- 285.

A.37. [CB-03] C. Boutilier, R. Das, J. Kephart, G. Tesauro, and W. Walsh, "Cooperative negotiation in autonomic systems using incremental utility elicitation", In *Nineteenth Conference on Uncertainty in Artificial Intelligence*, 2003, pp. 89–97.

A.38. [CH-01] C. V. Hollot, V. Misra, D. Towsley, and W. B. Gong, "A control theoretic analysis of RED", *In Proceedings of IEEE INFOCOM*, Anchorage, Alaska, USA 2001, pp. 1510–1519.

A.39. [CH-01a] C. V. Hollot, V. Misra, D. Towsley, and W. B. Gong, "On designing improved controllers for AQM routers supporting TCP flows", In *Proceedings of IEEE INFOCOM '01*, (Anchorage, Alaska), Vol.35, 2001, pp.1936-1950.

A.40. [CH-01b] C. V. Hollot, V. Misra, D. Towsley, and W. B. Gong, "A controltheoretic analysis of RED," in *Proceedings of IEEE INFOCOM'01*, (Anchorage, Alaska), Vol.35, 2001, pp. 1016-1038.

A.41. [CH-04] M.C.Huebscher and J.A. McCann, "Adaptive middleware for context-aware applications in smart-homes", In Proceedings of the 2nd Workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC), 2004, pp 215-222.

A.42. [CH-05] M.C.Huebscher and J.A. McCann, "Using real-time dependability in adaptive service selection", In *Proceedings of the International Conference on Autonomic and Autonomous Systems (ICAS)*. 76–81.IBM. 2003. An architectural blueprint for autonomic computing. Tech. rep., IBM, 2005

A.43. [CI-00] C. Intanagonwiwat, R. Govindan and D. Estrin, "Directed diffusion: A scalable and robust communication paradigm for sensor networks", In Proceedings of Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM), 2000, pp. 56–67.

A.44. [CL-00a] C. Lu, J. A. Stankovic, T. F. Abdelzaher, G. Tao, S. H. Son, and M. Markley. "Performance specifications and metrics for adaptive real-time systems", In *Proceedings of the IEEE Real Time Systems Symposium*, Orlando, 2000, pp.567-578.

A.45. [CL-00b] C. Lu, J. A. Stankovic, T. F. Abdelzaher, G. Tao, S. H. Son, and M.Markley, "Performance specifications and metrics for adaptive digital systems," In *Proceedings of the IEEE Real Time Systems Symposium*, (Orlando), 2000, pp.1137-1152.

A.46. [CL-05] C. Lu, X. Wang, and X. Koutsoukos, " Feedback utilization control in distributed real- time systems with end-to-end tasks", *IEEE Transactions on Parallel and Distributed Systems*, Nr. 16, 2005, pp.550–561.

A.47. [CM-05] C. Molina-Jimenez, J. Pruyne and A. van Moorsel, "The Role of Agreements in IT Management Software", Architecting Dependable Systems III, LNCS 3549, 2005, pp.117-128.

A.48. [CR-05] C. Roblee, V. Berk and G. Cybenko, "Implementing large-scale autonomic server monitoring using process query systems", In Proceedings of the 2nd International Conference on Autonomic Computing (ICAC), 2005, pp.123–133.

A.49. [CS-02] C. Santos, X. Zhu and H. Crowder, "A Mathematical Optimization Approach for Resource Allocation in Large Scale Data Centers,", *HP Labs Technical Report, HPL-2002-64R1,* HP Laboratories, USA 2002.

A.50. [DA-02] D.Aranson, R.Buyya and J.Giddy, "A computational economy for grid computing and its implementation in the Nimrod-G resource broker". *Future Gener. Comput. Syst. Nr.18*, vol.8, 2002, pp. 1061–1074.

A.51. [DA-05]D.Arawal, S.Calo, J.Giles, K.W.Lee and D.Verma, "Policy management for networked systems and applications", In Proceedings of the 9th IFIP/IEEE International Symposium on Integrated NetworkManagement,2005, pp. 455–468.

A.52. [DB-05] D. Breitgand, E. Hennis, and O. Shehory. "Automated and adaptive threshold setting: Enabling technology for autonomy and self-management", In *Proceedings of the Second International Conference on Autonomic Computing*. IEEE Computer Society, 2005, pp.414-427.

A.53. [DC-03]D. D. Chambliss, G. A. Alvarez, P. Pandey, D. Jadav, J. Xua, R. Menon, and T. P. Lee. "Performance virtualization for large-scale storage systems". In *Proceedings of the 22nd International Symposium on Reliable Distributed Systems*, IEEE Computer Society, 2003, pp. 109-118

A.54. [DC-04a] D. M. Chess, A. Segal, I. N. Whalley, and S. R. White. "Unity: Experiences with a prototype autonomic computing system", In *Proceedings of the First International Conference on Autonomic Computing*, IEEE Computer Society, 2004, pp. 140-147

A.55. [DC-04b] D. Chakeres and E.M. Belding-Royer, "Aodv routing protocol implementation design", In Proceedings of the 24th International Conference on Distributed Computing Systems Workshops (ICDCSW'04).IEEE Computer Society, USA, 2004, pp.698–703.

A.56. [DC-05]D. M. Chess, G. Pacifici, M. Spreitzer, M. Steinder, A. Tantawi, and I. N. Whalley. "Experience with collaborating managers: Node group manager and provisioning manager." In *Proceedings of the Second International Conference on Autonomic Computing*. IEEE Computer Society, 2005, pp. 1196-1208.

A.57. [DC-06] D. Chu, T. Chen, and H. Marquez, "Explicit robust model predictive control using recursive closed-loop prediction," *International Journal of Robust Control*, vol. 16, no. 11, 2006, pp. 519–546.

A.58. [DG-01] D. Garlan, B.Sshmerl and J. Chang, “Using gauges for architecture-based monitoring and adaptation”. In Working Conference on Complex and Dynamic Systems Architecture, Brisbane, Australia, 2001, pp.711-722.

A.59. [DG-02a] D.Garlan and B.Schmerl, „Exploiting architectural design knowledge to support self-repairing systems.”, In Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering, USA 2002, pp.547-559.

A.60. [DG-02b] D. Garlan and B. Schmerl, “Model-based adaptation for self-healing systems”, *Proceedingsof the first workshop on Self-healing systems*, ACM Press, Charleston, South Carolina, 2002, pp. 928-939

A.61. [DG-02c] D. Garlan, B.Schmerl, S.W.Cheg, J.Sousa, B.Spitznagel and P. Steenkiste, „Using architectural style as a basis for system self-repair”. In *Proceedings of the 3rd Working IEEE/IFIP Conferenceon Software Architecture*, USA 2002, pp.202-214.

A.62. [DH-06] D. Hirsch, J. Kramer, J. Magee and S. Uchitel, “*Models for Software Architectures*, *Third* European Workshop on Software Architecture” (EWSA 2006), Springer, Nantes, France, 2006, pp.555-568.

A.63. [DI-07] D. Ionescu, B. Solomon, M. Litoiu and M. Mihaescu, “An Autonomic Computing Approach to Server Virtualization”. 6th International Information and Telecommunication Technologies Symposium ,Cuiaba, Brazil 2007, pp.212-236

A.64. [DJ-02] D. Jackson “Alloy: a lightweight object modelling notation”, *Softw. Engin. Method. Nr.11*, Vol.2, 2002, pp. 256–290.

A.65. [DM-00a] D. Mayne, J. Rawlings, C. Rao, and P. Scokaert, “Constrained model predictive control: stability and optimality,” *Automatica*, vol. 36, no. 6, 2000, pp.590-601.

A.66. [DM-00b] D.A. Menasce, V.A.F. Almeida, R. Fonsece, and M.A. Mendes. “Business oriented resource management policies for e-commerce servers”, *Performance Evaluation*, nr. 42, vol. 2-3, 2000, pp. 223-239.

A.67. [DM-01] D. Menasce and V. A. F. Almeida. “Capacity Planning for Web Performance: Metrics, Models, and Methods”, Ed. Prentice Hall, NY USA 2001.

A.68. [DM-03] D. A. Menasce, M. Bennani, "On the Use of Performance Models to Design Self-Managing Computer Systems," *Proceedings Computer Measurement Group Conference*, Dallas, 2003, pp.201-220.

A.69. [DM-06] D. Munoz de la Pena, A. Bemporad, and C. Filippi, “Robust explicit MPC based on approximate multiparametric convex programming,” *IEEE Transactions on Automatic Control*, vol. 51, no. 8, 2006, pp. 1399 – 1403.

A.70. [EC-04] E.F Camacho and C. Bordons, “Model Predictiv Control”, second edition, Ed. Springer, Werlag, USA 2004.

A.71.[ED-02] E. M. Dashofy, A. van der Hoek and R. N. Taylor, “*Towards architecture-based self*-healing systems”, Proceedings of the first workshop on Self-healing systems, ACM Press, Charleston, South Carolina, USA 2002, pp.717-725.

A.72. [EE-02] E. N. Elnozahy,M. Kistler, and R. Rajamony. “Energy-efficient server clusters”, In Power Aware Computing Systems, USA 2002, pp.179-196.

A.73.[EF-05] E.Femal and V.W. Freeh, "Boosting data center performance through non-uniform power allocation", In Proceedings of the 2nd International Conference on Autonomic Computing (ICAC), USA 2005, pp.120-128.

A.74. [EM-02] E. Mainsah, "Autonomic computing: The next era of computing". *Electron. Comm. Engin. Journal, 14*, 1, 2–3, USA 2002, pp.20-30

A.75. [EM-03] E. Manoel, S. C. Brumfield, K. Converse, M. DuMont, L. Hand, G. Lilly, M. Moeller, A. Nemati, and A. Waisanen, ''Provisioning On Demand: Introducing IBM Tivoli Intelligent ThinkDynamic Orchestrator,'' IBM Redbooks, NY USA 2003; see http://www.redbooks.ibm.com/abstracts/sg248888.html.

A.76. [EM-05] E.Manoel, M.J. Nielsen, A.Salahsour and S.Sampath, "*Problem Determination Using* Self-Managing Autonomic Technology", IBM Redbooks, USA 2005, pp. 444-456.

A.77. [ET-05] E. Thereska, D. Narayanan, G. R. Ganger: "Towards self-predicting systems: What if you could ask "what-if"? 3rd International Workshop on self-adaptive and autonomic computing systems. Copenhagen, Denmark 2005, pp. 105-118.

A.78. [EW-02] E.Wieselthier, G.D.Nguyen and A. Ephremides, „Energy-efficient broadcast and multicast trees in wireless networks,. *Mob. Netw. Appl. Nr. 7*, Vol.6, 2002, pp. 481–492.

A.79. [EW-04] E.Walsh, G. Tesauro, J.O. Kephart and R. Das, "Utility functions in autonomic systems", In Proceedings of the 1st International Conference on Autonomic Computing (ICAC'04), USA 2004, pp. 70–77.

A.80. [FB-03] F. Bantz, C. Bisdikian, D.Challener, J.P. Karidis, S.Mastrianni, A. Mohindra., D.G Shea and M.Vanover, "Autonomic personal computing", *IBM Syst. Journal nr.42*, Vol.1, USA 2003, pp. 165–176.

A.81. [FC-02] F. Cuzzola, J. Geromel, and M. Morari, "An improved approach for constrained robust model predictive control," *Automatica*, vol. 38, no. 7, USA 2002, pp. 1183 – 1189.

A.82. [FK-00] F. Kon, R.H. Campbell, M.D. Mickunas, K.Nahrstedt and F.J.Ballesteros, "A distributed operating system for dynamic heterogeneous environments". In Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing, USA 2000, pp. 201–210.

A.83. [FS-04] F. Saffre and H. R. Blok. SelfService: "A theoretical protocol for autonomic distribution of services in P2P communities". In *Proceedings of the First International Conference on Autonomic Computing*, IEEE Computer Society, USA 2004, pp.326-337.

A.84. [GC-03] G. Candea, E.Kiciman, S.Zhang, P.Keyani and A. Fox, "An autonomous self-recovering application server". In Proceedings of the Autonomic Computing Workshop, USA 2003, pp. 168–177.

A.85. [GC-04] G. Candea, S. Kawamoto, Y. Fujiki, G. Friedman, and A. Fox, "Microreboot—A technique for cheap recovery". In Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI), USA 2004, pp.215-244.

A.86. [GG-05] G. C. Goodwin, M. M. Seron, and J. A. De Dona, "*Constrained Control and Estimation : An Optimisation Approach"*, 1st Ed. Springer Verlag, NY USA 2005.

A.87 [GH-06] G.Handsen, E. Christiansen and E. Jul, "The laundromat model for autonomic cluster computing". In Proceedings of 3rd IEEE International Conference on Autonomic Computing (ICAC), USA 2006, pp. 672-681.

A.88. [GK-00] G. Kaiser and G. Valetto, "Ravages of time: Synchronized multimedia for internet-wide processcentered software engineering environments", In Proceedings of the 3rd ICSE Workshop on Software Engineering over the Internet, USA 2000, pp.555-566.

A.89. [GK-03a] G.Kaiser, P.Gross, J. Parekh and G.Valetto, "An approach to autonomizing legacy systems". In Proceedings of the Workshop on Self-Healing, Adaptive and Self-Managed Systems, USA 2003, pp. 202-214.

A.90.[GK-03b] G.Kaiser, J. Parenkh, P.Gross and G. Valetto, "Kinesthetics extreme: An external infrastructure for monitoring distributed legacy systems", In Proceedings of the Autonomic Computing Workshop at the 5th Annual International Workshop on Active Middleware Services (AMS), USA 2003, pp. 857-865.

A.91. [GK-05] G. Koloniari and E. Pitoura, "Peer-to-Peer Management of XML Data: Issues and Research Challenges," *SIGMOD Record*, vol. 34, USA 2005, pp. 6–17.

A.92. [GS-03]G. D. M. Serugendo, A. Karageorgos, O. F. Rana, and F. Zambonelli, editors. "*Engineering Self-Organising Systems"*. Ed. Springer-Verlag, Berlin, 2003.

A.93. [GT-04] G. Tesauro, D. M. Chess, W. E. Walsh, R. Das, A. Segal, I. Whalley, J. O. Kephart, and S. R. White. "A multi-agent systems approach to autonomic computing", In AAMAS, IEEE Computer Society, USA 2004, pp. 464-471.

A.94. [GT-05] G. Tesauro, R. Das, W. E. Walsh, and J. O. Kephart. "Utility-function-driven resource allocation in autonomic systems", In Second International Conference on Autonomic Computing, NJ USA 2005, pp. 359-368.

A.95. [GT-06] G.Thomson, G. Stevenson, S.Terzis and P.Nixon, "A self-managing infrastructure for ad-hoc situation determination". In 4th International Conference on Smart Homes and Health Telematics (ICOST'06). IOS Press USA 2006, pp.352-360.

A.96. [GV-02] G.Valetto and G. Kaiser, "A case study in software adaptation". In *Proceedings of the* 1st Workshop on Self-Healing Systems, USA 2002, pp. 73–78.

A.97. [GV-03] G. Valetto and G.Kaiser, "Using process technology to control and coordinate software adaptation". In Proceedings of the 25th International Conference on Software Engineering (ICSE'03). IEEE Computer Society, USA 2003, 262–272.

A.98.[HC-00] H. Carstea, "Constructia si Tehnologia Echipamentelor Electronice" , Ed. Politehnica, Timisoara 2000, ISBN: 973-9389-54-6.

A.99. [HC-02] H. Cârstea, R.Mihăescu," A. Method for Optimal Partitioning Digital Systems using Redundancy Techniques", Buletinul ştiinţific al Univ. "POLITEHNICA", Tom 43, Fascicula 1-2, Timişoara 2002, pp.205-208

A.100. [HC-07] H.Cârstea, "Strategii de Inspecţie şi Testare în Electronică", Ed. De Vest, Timisoara 2007, ISBN: 978-973-36-0457-0.

A.101. [HC-08a] H. Cârstea,D. Margeloiu,O. Mitaru, "Redundancy and Testability in Digital Filters", *Bul. Stiint. Univ. Politehnica Tm.*, Ser. *Electronica si TC, Tomul 53(67),Fascicola 1, Timisoara 2008, pp. 204-206*

A.102. [HC-08b]H. Cârstea, O. Mitaru,R. Negrea, "Method for Computing a Reliability Function of Digital Systems with Redundant Structure Through Coding", *ISSE 2008, 31st International Spring Seminar on Electronics Technology, Budapest, Hungary 2008, pp.48-53*

A.103. [HF-05] H. Fukushima and R. R. Bitmead, "Robust constrained predictive control using comparison model," *Automatica*, vol. 41, no. 1, USA 2005,pp. 97 – 106.

A.104.[HF-06] H. Foster, J. Magee, S. Uchitel and J. Kramer, "*Scenario- Based Software Synthesis for* Adaptable Software Architectures of UAVs", Proceedings of First Annual SEAS DTC Conference, www.seasdtc.com, Edinburgh, 2006, pp.416-426.

A.105. [HG-03a] H.Guo, "A bayesian approach for autonomic algorithm selection:. In *Proceedings of the* IJCAI Workshop on AI and Autonomic Computing: Developing a Research Agenda for Self-Managing Computer Systems. Acapulco, Mexico 2003, pp.332-342.

A.106. [HG-03b] H. Gomaa and M. Hussein, "*Dynamic Software Reconfiguration in Software Product* Families", 5th International Workshop on Software Product-Family Engineering, LNCS 3014, Springer, Siena, Italy 2003, pp.435-444.

A.107. [HK-05a] H.Kamoda, M.Yamaoka, S.Matsuda, K.Broda and M.Sloman, "Policy conflict analysis using free variable tableaux for access control in web services environments", In Proceedings of the PolicyManagement for the Web Workshop at the 14th International World Wide Web Conference (WWW), USA 2005, pp.672-684.

A.108. [HK-05] H. Kreger and T. Studwell, "Autonomic Computing and Web Services Distributed Management," IBM white paper, USA 2005;
http://www.ibm.com/developerworks/autonomic/library/ac-architect/

A.109. [HL-01] H. Lutfiyya, G.Molenkamp, M.Katchabaw and M.A. Bauer, "Issues in managing soft qos requirements in distributed systems using a policy-based framework", In *Proceedings of the International Workshop on Policies for Distributed Systems and Networks (POLICY'01)*. Ed. Springer-Verlag, Berlin 2001,pp. 185–201.

A.110. [HL-02] H. Ludwig, A. Keller, A. Dan and R. King. "*A Service Level Agreement Language for Dynamic Electronic Services"*. Proceedings of the 4th IEEE Int'l Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS '02), USA 2002, pp. 96-104.

A.111. [HL-04] H. Liu, M. Parashar and S. Hariri. "A Component Based Programming Framework for Autonomic Applications", *ICAC'04*, IEEE, New York, 2004, pp. 10-17.

A.112. [IA-07] I. Aib and R. Boutaba, "Business-Driven Optimization of Policy-Based Management Solutions", 10th IFIP/IEEE International Symposium on Integrated Network Management (IBM), USA 2007, pp.551-569.

A.113. [IG-02] I. Georgiadis, J. Magee and J. Kramer, "*Self-organising software architectures for* distributed systems", Proceedings of the first workshop on Self-healing systems, Ed. ACM Press, Charleston, South Carolina, USA 2002, pp.1007-1018.

A.114. [IK-00] I.Karsligil, "Multi-scale modeling and model predictive control of processing systems" PhD thesis, Massachusetts Institute of Technology, USA 2000.

A.115. [JB-02] J. P. Bigus, D. A. Schlosnagle, J. R. Pilgrim, W. N. Mills III and Y. Diao, "ABLE: A toolkit for building multiagent autonomic systems", IBM Systems Journal, Vol 41, No 3, USA 2002, pp.41-59.

A.116. [JB-04a] J. S. Bradbury, J. R. Cordy, J. Dingel and M. Wermelinger, "*A survey of self-management in* dynamic software architecture specifications", Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems, Ed. ACM Press, Newport Beach, California, USA 2004,pp. 917-928.

A.117. [JB-04B] J.Burrell, T.Brooke and R. Beckwith, :Vineyard computing: Sensor networks in agricultural production". *IEEE Pervasive Comput.Nr. 3*, Vol.1, USA 2004, pp.38–45.

A.118. [JC-01] J.M.C. Clark, R.B. Winter: "Flow rate control," Electronics letters vol. 36, No.15, NY, J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle, "Managing Energy and Server Resources in Hosting Centers," In Proceedings of the ACM Symposium on Operating System Principles, New York, NY, USA, 2001,pp. 103–116.

A.119. [JD-06] J.Dowling, E.Curran, R.Cunningham and V.Cahill, "Building autonomic systems using collaborative reinforcement learning". *Knowle. Engin. Rev. Journal.* Special Issue on Autonomic Computing. Cambridge University Press, USA 2006, pp.551-562.

A.120. [JG-04] J. Gao, G. Kar, and P. Kermani. "Approaches to building self healing systems using dependency analysis". In Proceedings of IEEE/IFIP Network Operations and Management Symposium (NOMS), NY, USA 2004, pp.82-96.

A.121. [JH-04] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury." *Feedback Control of Computing Systems"*, Ed. John Wiley & Sons, NY, USA 2004.

A.122. [JK-00]J. O. Kephart and G. J. Tesauro." Pseudo-convergent Q-learning by competitive pricebots. "In Proceedings of the Seventeenth International Conference on Machine Learning (ICML'00), Stanford, CA, USA 2000, pp.463-470.

A.123. [JK-01] J. O. Kephart, C. H. Brooks, R. Das, J. K. MacKie-Mason, R. S. Gazzale, and E. H. Durfee. "Pricing information bundles in a dynamic environment", In *Proceedings of ACM EC-01*, USA 2001, pp.198-202.

A.124. [JK-05] J. O. Kepart and D. M. Chess, T. Kimbrel, M. Steinder, M. Sviridenko, and A. Tantawi. „Dynamic application placement underservice and memory constraints", In 4th International Workshop on Efficient and Experimental Algorithms, Santorini Island, Greece 2005, pp.198-204.

A.125. [JK-04] J. O. Kephart and W. E. Walsh. An artificial intelligence perspective on autonomic computing policies. In *Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks*, IEEE Computer Society, Los Alamitos, USA 2004, pp.3-12.

A.126. [JM-02] J. M. Maciejowski: "*Predictive Control with Constraints"*, Ed. Prentice Hall, Englewood Cliffs, NJ,USA 2002.

A.127. [JM-06] J.Moore, J.Chase and P. Ranganathan, "Automated, online, and predictive thermal mapping and management for data centers", In *Proceedings of the 2nd International Conference on Autonomic Computing (ICAC)*. Dublin, Ireland 2006,pp. 155–164.

A.128. [JP-03] J.Parekh, G.Kaiser, P.Gross and G. VAletto, "Retrofitting autonomic capabilities onto legacy systems:. Tech. rep. CUCS-026-03, Columbia University, USA 2003.

A.129. [JR-02] J.Rutherford,K. Anderson,A. Carzaniga,D. Heimbigner and A.L.Wolf, " Reconfiguration in the Enterprise Javabean component model". In *Proceedings of the IFIP/ACM Working Conference on Component Deployment*, Berlin, Germany 2002, pp. 67–81.

A.130. [JW-05] J.Wildstrom, P.Stome, E.Witchel, R.Mooney and M. Dahlim," Towards self-configuring hardware for distributed computer systems. In Proceedings of the 2nd International Conference on Autonomic Computing (ICAC), Olando USA 2005, pp. 241–249.

A.131. [JX-05] J.Xu, S.Adabala and J.A. Fortes," Towards autonomic virtual applications in the in vigosystem". In Proceedings of the 2nd International Conference on Autonomic Computing (ICAC), Orlando USA 2005, pp.412-428.

A.132. [JZ-06]J. Zhang and B. H. C. Cheng, "Model-based development of dynamically adaptive software", Proceeding of the 28th international conference on Software engineering, Ed. ACM Press, Shanghai, China, 2006, pp. 712-726.

A.133. [JZ-07] J. Zhang and R. Figueiredo, " Autonomic feature selection for application classification", In Proceedings of the International Conference on Autonomic Computing (ICAC 07). Orlando USA 2007,pp. 555-566.

A.134. [KA-01] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D. Pazel, J. Pershing, and B. Rochwerger, "Océano - SLA Based Management of a Computing Utility," In Proceedings of the 7th IFIP/IEEE International Symposium on Integrated Network Management, Seattle, WA, USA 2001, pp. 64-71.

A.135. [KA-06] K. Astrom. "Challenges in Control Education". *Advances in Control Education*, Ed.Academic Press, NY, USA 2006.

A.136. [KB-03] K. P. Birman, R. van Renesse, and W. Vogels. "Navigating in the storm: Using Astrolabe for distributed self-configuration, monitoring and adaptation". In *Proceedings of the 5th Annual International Workshop on Active Middleware Services*, 2003,pp.78-89.

A.137. [KB-04] K. P. Birman and R. M. Saikat Guha. "Scalable, self-organizing technology for sensor networks". In B. Yeler, editor, *Advances in Pervasive Computing and Networking*. Ed. Kluwer Academic Press, USA 2004.

A.138. [KC-05] K. Czajkowski, I. Foster, and C. Kesselman, "Agreement-Based Resource Management," *Proc. IEEE*, vol. 93, 2005, pp. 631–643.

A.139. [KF-05] K. Fujii and T. Suda, "Semantics-Based Dynamic Service Composition," *IEEE JSAC*, vol. 23, 2005, pp. 2361–2372.

A.140. [KG-03] K. Gopalan: "Efficient Provisioning Algorithms for Network Resource Virtualization with QoS Guarantee", Ph.D. thesis, Ontario, Canada 2003.

A.141. [KK-02] K. Kanoun, H. Madeira, and J. Arlat. "A framework for dependability benchmarking". In *Proceedings of the Workshop on Dependability Benchmarking (DSN 2002)*, NY, USA 2002.

A.142. [KK-07] K. Kontoginannis , G. Lewis, D. Smith, M. Litoiu, H. Muller, S.Schuster, E.Stroulia, "The Landscape of Service-Oriented Systems, A Research Perspective," Proceedings of SDSOA, Mineappolis, USA 2007.

A.143. [KL-01] K. Li, M. H. Shor, J.Walpole, C. Pu, and D. C. Steere, "Modeling the effect of short-term rate variations on tcp-friendly congestion control behavior," in *Proceedings of the American Control Conference*, USA 2001, pp. 3006–3012.

A.144. [KS-01] K. Sycara. "Multi-agent infrastructure, agent discovery, and middle agents for web services and interoperation". In Multi-agents systems and applications, Springer-Verlag New York, Inc., NY, USA 2001, pp. 17-49.

A.145. [KS-02] K. Shen, H. Tang, T. Yang, and L. Chu, "Integrated Resource Management for Cluster-Based Internet Services," In Proceedings of the 5th Symposium on Operating Systems Design and Implementation, Boston, MA, USA 2002, pp. 225-238.

A.146. [LB-04]L. Baresi, C. Ghezzi and S. Guinea," *Smart monitors for composed services"*, Proceedings of the 2nd international conference on Service oriented computing, Ed. ACM Press, New York, NY, USA, 2004.

A.147. [LC-00]Chenyang Lu, Jack A. Stankovic, Tarek F. Abdelzaher, Gang Tao, S.H. Son, and M. Marley. "Performance specifications and metrics for adaptive real time systems". In *Proceedings 21st IEEE Real Time Systems Symposium*, USA 2000, pp.13-24.

A.148. [LH-04] L.Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. "Feedback Control of Computing Systems",Ed. Wiley Inter-Science, USA 2004. Also see http://www.research.ibm.com/fbcs/.

A.149. [LL-03] L.Lymberopoulos, E.Lupu and M.Sloman."An adaptive policy-based framework for network services management". *J. Netw. Syst. Manag. 11*, (Special Issue on Policy-based Management), 2003.

A.150. [LS-02a] L. Sha, X. Liu, Y. Lu, and T. Abdelzaher, "Queueing model based network server performance control," in *IEEE Real-Time Systems Symposium*, 2002, pp.482-298.

A.151. [LS-02b] L.Sha, X. Liu, Y. Lu, and T. Abdelzaher. "Queueing Model Based Network Server Performance Control". IEEE RealTime Systems Symposium, USA 2002, pp.16-32.

A.152. [LS-02c]L. Sha, X. Liu, Y. Lu, T. F. Abdelzaher: "Queuing model based network server performance control," IEEE Real-Time Systems Symposium, LA, USA 2002, pp.96-102.

A.153. [MM-03] M.Mihaescu, A Trossman, "Incresing the Availability of Complex Systems through Dinamic Automate Reconfiguration", In Proceedings of the International Symposium for Design and Technology of Electronic Packages(SIITME-03), Timisoara 2003, pp.235-237.

A.154. [PL-01] P.K.Lala, "Self-Checking and Fault-Tolerant Digital Design", Ed. Academic Press Morgan Kaufmann Publisher, London UK 2001

A.155. [TA-02] T. Abdelzaher, K.J. Shin and N. Bhatti, "Performance Guarantees for Web Server End-Systems: A Control- Theoretical Approach". *IEEE Transactions on Parallel and Distributed Systems*, Vol. 13, No. 1,USA 2002, pp.1354-1368.

A.156. [TE-03] T. Eymann, M. Reinicke, O. Ardaiz, P. Artigas, F. Freitag, L. Navarro. "Self-Organizing Resource Allocation for Autonomic Networks.", *DEXA 2003*, Prague, Czech Republic. IEEE Computer Society Proceedings,Prague, Czech Republic 2003, pp. 656–660.

A.157. [TE-04] T. Eilam, M. Kalantar, A. Konstantinou, and G. Pacifici. "Model-based automation of service deployment in a constrained environment". Technical report, IBM, RC23382, USA 2004.

A.158. [TH-03] T. Hayakaway: "Direct Adaptive Control for Nonlinear Uncertain Dynamical Systems" Ph.D. Thesis, Georgia Institute of Technology, 2003

A.159. [TK-03] T. Kelly. "Utility-directed allocation". In *Proceedings of the First Workshop on Algorithms* and Architectures for Self-Managing Systems, LA, USA 2003, pp.299-315.

A.160. [TZ-05] T. Zheng, J.Yang, M.Woodside, M.Litoiu, G. Islzai."Tracking Time-Varying Parameters in Software Systems with Extended Kalman Filters," Proceedings of CASCON 05, ACM Digital Library, USA 2005,pp.459-472.

A.161. [TZ-06] T.Zenmyo, H.Yoshida and T.Kimura, "A self-healing technique based on encapsulated operation knowledge". In Proceedings of 3rd IEEE International Conference on Autonomic Computing (ICAC). Dublin, Ireland 2006, pp.25–32.

A.162. [VH-01] V.C.Hollot, V. Misra, D. Towsley, and W.B. Gong. "A Control Theoretic Analysis of RED". Proceedings of IEEE INFOCOM '01, USA 2001, pp.1567-1581.

A.163. [VM-00] V. Misra, Bo Gong Wei and D. Towsley. "Fluid based analysis of a network of AQM routers supporting TCP flows with an application to RED". In *ACM SIGCOMM*, USA 2000, pp. 151–160.

A.164. [VM-03] V.Markl, G.Lohman and V. Raman, "LEO: An autonomic query optimizer for DB2". *IBM Syst. Journal, Nr.42*, Vol.1,USA 2003, pp.98–106.

A.165. [VS-03] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and Z. Lu. "Power-aware QoS management in Web servers". In *RTSS '03: Proceedings of the 24th IEEE International Real-Time Systems Symposium*, IEEE Computer Society, Washimgton DC, USA 2003, pp.63-86.

A.166. [WC-02] W.Chenglin , Z. Donghua Zhou. "Multiscale Estimation Theory and Its Application".Ed. Tsinghua University Press, Beijing 2002.

A.167. [WH-1] W. M. Haddady, V.S.Chellaboinaz, T. Hayakaway: "Robust Adaptive Control for Nonlinear Uncertain Systems" Proc. Of the 40-th IEEE CDC, Orlando, USA 2001, pp.1615-1620.

A.168. [WK-03] W. Katsurashima, S. Yamakawa, T. Torii, J. Ishikawa, K. Yamaguti, K. Fujii, T. Nakashima: "NAS Switch: A Novel CIFS Server Virtualization". Proceedings. 20th IEEE/11th NASA Goddard Conference on Mass Storage Systms and Technologies, 2003. (MSST 2003), pp.153-169.

A.169. [WS-05] W.J.Strickland, V.W.Freeh, X.Ma and S.S.Vazhkudai, "Governor: Autonomic throttling for aggressive idle resource scavenging". In Proceedings of the 2nd International Conference on Autonomic Computing (ICAC), USA 2005, pp. 117-130.

A.170. [WS-07] W. Schaefer and H. Wehrheim, "The Challenges of Building Advanced Mechatronic *Systems"*, in L. Briand and A. L. Wolf, eds., *Future of Software Engineering 2007*, IEEE, Ed. CS Press, 2007.

A.171. [WW-04] W. E. Walsh, G. Tesauro, J. O. Kephart, and R. Das. "Utility functions in autonomic systems". In Proceedings of the First International Conference on Autonomic Computing, IEEE Computer Society, 2004, pp.70-77.

A.172. [WX-06] W. Xu, X. Zhu, S. Singhal, and Z.Wang. "Predictive control for dynamic resource allocation in enterprise data centers". In Proceedings of the IEEE/IFIP Network Operations & Management Symposium, USA 2006, pp.458-470.

A.173. [WY-03] W. Yuan and K. Nahrstedt. "Energy-efficient soft real-time cpu scheduling for mobile multimedia systems". In SOSP '03: Proceedings of the 19th ACM Symposium on Operating Systems Principles, Ed. ACM Press, NY, USA, 2003, pp.149-163.

A.174. [XD-03] X. Dong, S. Hariri, L. Xue, H. Chen, M. Zhang, S. Pavuluri, S. Rao. "Autonomia: an autonomic computing environment,," IEEE International Conference on Performance, Computing, and Communications, USA 2003, pp. 61-68.

A.175. [XL-07] X. Liu, X. Zhu, P. Padala, Z. Wang, and S. Singhal. "Optimal multivariate control for differentiated services on a shared hosting platform". In *Proceedings of the IEEE Conference on Decision and Control*, USA 2007,pp.997-1009.

A.176. [XW-06] W. Xu, X. Zhu, S. Singhal, Zhikui Wang:" Predictive Control for Dynamic Resource Allocation in Enterprise Data Centers" NOMS 2006, Vancouver, 2006, pp. 118-126

A.177. [XW-07] X. Wang, Y. Chen, C. Lu, and X. Koutsoukos. „FC-ORB: A robust distributed real-time embedded middleware with end-to-end utilization control". *Journal of Systems and Software*, No.80(7), USA 2007, pp.938–950.

A.178. [XZ-06] X. Zhu, Z. Wang, and S. Singhal. „Utility driven workload management using nested control design". In Proceedings of the American Control Conference, USA 2006, pp. 1152-1168.

A.179. [YD-02a] Y. Diao, N. Ghandi, J. L. Hellerstein, S. Parekh, D. Tilbury: "Using MIMO feedback control to enforce policies for interrelated metrics with application to the Apache Web server," IEEE/IFIP Network Operations and Management, Vol. 7, Nr.3, USA 2002, pp.245-251.

A.180. [YD-02b]Y. Diao, N. Gandhi, J.L. Hellerstein, S. Parekh, and D.M. Tilbury, "MIMO control of an Apache Web server: Modeling and controller design," American Control Conference, 2002, pp.78-89.

A.181. [YD-02c]Y. Diao, J. L. Hellerstein, and S. Parekh, "Optimizing quality of service using fuzzy control," in IEEE Distributed Systems Operations and Management, 2002, pp.1151-1169.

A.182. [YD-03] Y.Diao, X. Lui, S.Froehlich, J.L.Hellerstein, S.Parekh and L.Sha, "On-Line Response Time Optimization of An Apache Web Server". *International Workshop on Quality of Service*, USA 2003,pp.498-517.

A.183. [YD-04b]Y. Diao, J. L. Hellerstein, A. Storm, M. Surendra, S. Lightstone, S. Parekh, and C. Garcia-Arellano. "Using MIMO linear control for load balancing in computing systems". In *Proceedings of the American Control Conference*, USA 2004, pp.2045-2050.

A.184. [YD-04c] Y. Diao, J. L. Hellerstein, A. J. Storm, M. Surendra, S. Lightstone, S. Parekh, and C. Garcia-Arellano. "Incorporating cost of control into the design of a load balancing controller". In *Proceedings of the Real-Time and Embedded Technology and Application Systems Symposium, Toronto, Canada 2004*, pp. 376–387.

A.185. [YD-05a]Y. Diao, C. W. Wu, J. L. Hellerstein, A. J. Storm, M. Surendra, S. Lightstone, S. Parekh, C.Garcia-Arellano,M.Carroll,L.Chu and J. Colaco. "Comparative studies of load balancing with control and optimization techniques". In *Proceedings of the American Control Conference, Portland, OR 2005*, pp. 1484–1490.

A.186. [YD-05b]Y.Diao, J.L.Hellerstein, S. Parekh, R.Griffith, G.Kaiser and D. Phung, „Self-managing systems: A control theory foundation". In Proceedings of the 12th IEEE

International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS), 2005

A.187. [YL-00] Y. Lu and Y. Arkun, "Quasi-min-max MPC algorithms for LPV systems," *Automatica*, vol. 36, no. 4, USA 2000, pp. 527 – 540.

A.188. [YL-01] Y. Lu, A. Saxena, T. F. Abdelzaher: "Differentiated caching services: A control-theoretic approach," International Conference on Distributed Computing Systems, Hawai 2001, pp. 78-84.

A.189. [ZW-03] Z. Wan and M. Kothare, "An efficient off-line formulation of robust model predictive control using linear matrix inequalities," *Automatica*, vol. 39, no. 5, USA 2003, pp. 837 – 846.

A.190. [ZW-05] Z. Wang, X. Zhu, and S. Singhal, "Utilization and SLO-based control for dynamic sizing of resource partitions," 16th IFIP IEEE Distributed Systems: Operations and Management, USA 2005, pp.1356-1379.

B. Platforme web on-line ale unor Societati cu preocupari in domeniul Sistemelor Autonome Dinamice

B.1. [***] http://researghwed.watson.ibm.com/autonomic/manifesto/autonomic_computing.pdf
B.2. [***] http://www.uddi.org.org/
B.3. [***] http://java.sun.com/javaee/technologies/
B.4. [***] http://www.oasis-open.org/committes/tc_home.php?wg_abbrev=wsdm
B.5. [***] http://www.eclipse.org/ptp/
B.6. [***] http://ws.apache.org/muse/
B.7. [***] http://publib.boulder.ibm.com/infocenter/tvhelp/v3r1/index.jsp?topic=/ com.ibm. tivoli. tio.
B.8. [***] http://www.-306.ibm.com/autonomic/pdfs/ACwpFinal.pdf
B.9. [***] http://www.nrl.navy.nil/aic/iss/aas/index.php
B.10. [***] http://www.w3.org/TRwsdl
B.11. [***] http://www.-306.ibm.com/software/tivoli/products/inteli-orch/
B.12. [***] http://java.sun.com/docs/boocs/tutorial/reflect/index.html
B.13. [***] http://www.-03.ibm.com/autonomic/openstandards.eshtml
B.14. [***] http://www.alphaworks.ibm.com/demo/flash/display/pmac0
B.15. [***] http://www.-106.ibm.com/developerworks/probdet.html
B.16. [***] http://www.sharcnet.ca
B.17. [***] http://xml.coverpages.org/SRMLsimpleRules-dtd.html
B.18. [***] http://www.w3.org/RDF
B.19. [***] http://www.hp.com/products1/promos/adaptiveenterprise/pdfs/visionforae.pdf
B.20. [***] http://www.ibm.com/security/cisco
B.21. [***] http://www.retbocs.ibm.com/redbooks/SG246635

C. Lucrări publicate de Drd.Ing.Mircea Mihăescu

C.1. [BS-06a] B.Solomon, D.Ionescu, M.Litoiu, **M.Mihaescu**: "Descentralized Predictive Control of Autonomic Computing Environments" 4th International Informatioin and Telecommunication Technologies Symposium (I2ts'2006), pp.94-103, Cuiaba, Brazil, December 6-8, 2006

C.2. [BS-07] B.Solomon, D.Ionescu, M.Litoiu, **M.Mihaescu**:"Toward a Real-Time Reference Arhitecture for Autonomic Systems" 29th International Conference on Software Engineering and Workshops(ICSE-ICSE Workshops 2007), pp 1224 – 1231.

C.3. [BS-07a] B.Solomon, D.Ionescu, M.Litoiu, **M.Mihaescu**, "A Real-Time Pattern Based Approch to Autonomic Computing" SEAMS 2007 ACM Workshop on Software Engineering for Adaptive and Self-Managing Systems, May 26-27, Minneapolis, Minnesota, USA, pp 78 – 85.

C.4. [BS-07b] B.Solomon, D.Ionescu, M.Litoiu, **M.Mihaescu**: "Adaptive Autonomic Computing", CASCON 2007, Toronto, Canada, October 23-25, 2007, pp 301 – 306.

C.5. [BS-07c] B. Solomon, D. Ionescu, M. Litoiu, and M. Mihaescu, "An autonomic computing approach to server virtualization", 5th International Information and Telecommunication Technologies Symposium, Ontario 2007, pp. 87-94.

C.6. [BS-07d] B. Solomon, D. Ionescu, M.Litoiu and M. Mihaescu, "A real-time adaptive control of autonomic computing environments". In Proceedings of the 2007 Conference of the Center For Advanced Studies on Collaborative Research (Richmond Hill, Ontario, Canada, 2007). NY,USA 2007, pp. 124-136.
DOI=http://doi.acm.org/10.1145/1321211.1321225

C.7. [DI-07] D.Ionescu, B.Solomon, M.Litoiu, **M.Mihaescu**:"An Autonomic Computing Approach to Server Virtualization" 6th International Informational and Telecommunication Technologies Symposium, Cuidaba, Brasil, pp 222 – 236.

C.8. [DI-08] D.Ionescu, B.Solomon, M.Litoiu, **M.Mihaescu:"** A Robust Autonomic Computing Arhitecture for Server Virtualization" IEEE-IINES 2008, 12th International Conference on Intelligent Engineering Systems, Miami, Florida, February 25-29, 2008, pp 127 – 136.

C.9. [ML-08] M.Litoiu, **M.Mihaescu**, B.Solomon, D.Ionescu:"Scalable Adaptive Web Services", In Proceeding of SDSOA 2008, Leipzig, May 10-18, 2008, pp 1107 – 1112.

C.10. [BS-08a] B.Solomon, D.Ionescu, M.Litoiu, **M.Mihaescu**: "Model-Driven Engineering for Autonomic Provisioned Systems ", MDDAS 2008, Turku, Finland, July 28-August 1, 2008, pp 21 – 29.

C.11. [BS-08b] B.Solomon, D.Ionescu, M.Litoiu, **M.Mihaescu**: "Web Service Distributed Management Framework for Autonomic Server Virtualization ", SVM08, Munich, Germany, October 21-22, 2008, pp 78 – 86.

C.12. [MM-03] **M.Mihaescu**, A.Trossman: "Increasing the Availability of Complex Systems through Dynamic Automatic Reconfiguration ", Proceedings of the 9th International Symposium for Design and Technology of Electronic Packages(SIITME), Timisoara, September 19, 2003, pp 235 – 239.

C.13. [MM-09a] **M.Mihaescu**: "Strategii de Crestere a Disponibilitatii Sistemelor Digitale Complexe". Ed.Politehnica, Timisoara 2009

C.14. [HC-09.a] H.Carstea, **M.Mihaescu**, L.Mihaescu: "Structuri Redundante Protective utilizand Reconfigurarea Autonoma Dinamica". Ed.AlmaMater, Sibiu 2009

C.15. [HC-09.b] H.Carstea, **M.Mihaescu**, L.Mihaescu: "Signal Detection from Noise Using Non Parametric Tests".The 17th Annual International Conference (ICCE 09-17), Hawaii, USA 2009 acceptata spre publicare – http://myweb.polyu.edu.hk/~mmktlau/ICCE/ICCE_Main.htm

C.16. [MM-09.c]**M. Mihaescu**, H.Carstea, L.Mihaescu: "Optimization Strategies for Multiple Decisions". The 17th Annual International Conference (ICCE 09-17), Hawaii, USA 2009 acceptata spre publicare http://myweb.polyu.edu.hk/~mmktlau/ICCE/ICCE_Main.htm

C.17. [TA-08] **M. Mihaescu**, A. Trossman et al: "Dynamic Configuration and Allocation of Resources in a Data Center". Brevet de inventator Nr 7308687/2007, Toronto, Canada http://www.patentgenius.com/patent/7308687.html

Drd. Ing. Mircea Florin Mihăescu is Director of Web-Ahead, Development and Security Innovations at IBM, Somers, NY, USA

- Director, Technology Innovations

IBM CIO Office, Somers NY – 2007 -- present

- Director, Provisioning and Configuration Technologies

IBM Tivoli, Toronto 2003 – 2007

- Vice President, Engineering

Think Dynamics, Toronto 2001 – 2003

- Vice President and Chief Technology Officer

beMany.com and energy.com, Toronto and New York 1999 – 2001

- Technology Head

Bank of Montreal Capital Markets, Toronto 1995 – 1999

- Senior Software Analyst

ALCATEL Canada Inc., Toronto 1995 – 1995

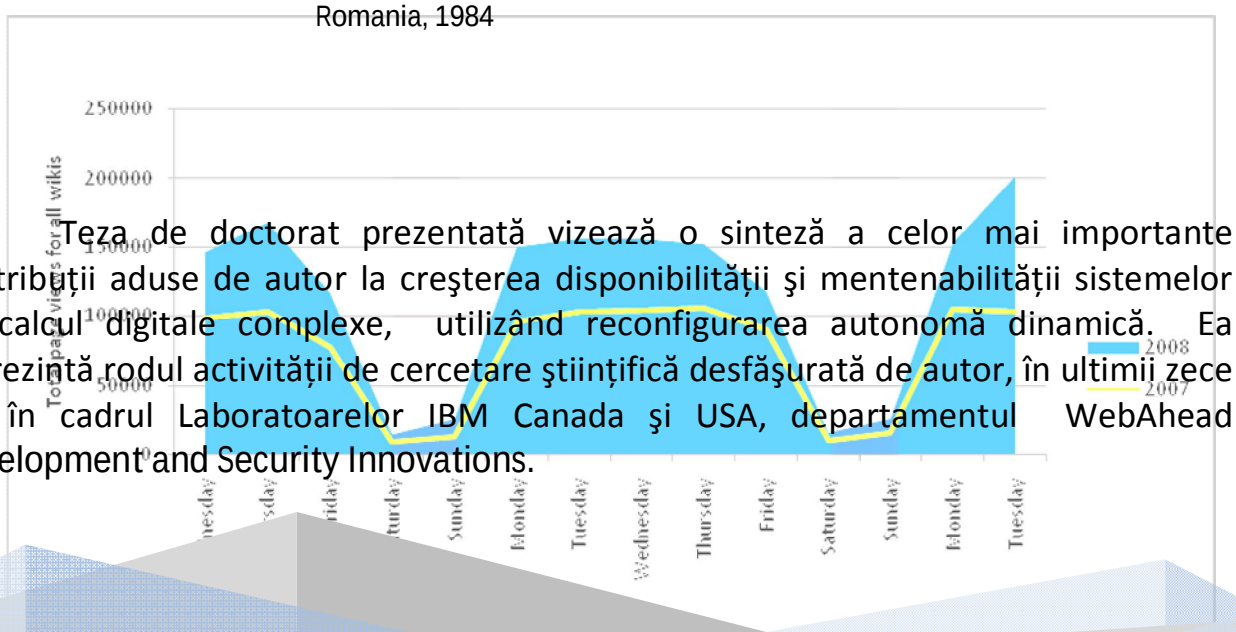
- Software Development Consultant

Mark Winter & Associates, Toronto 1993 – 1995

- Associate Professor of Control Systems

"Politehnica" University of Timisoara, Romania 1988 – 1992

M.Sc. Electronics Engineering from "Politehnica" University of Timisoara, Romania, 1984



Teza de doctorat prezentată vizează o sinteză a celor mai importante contribuții aduse de autor la creşterea disponibilității şi mentenabilității sistemelor de calcul digitale complexe, utilizând reconfigurarea autonomă dinamică. Ea reprezintă rodul activității de cercetare ştiințifică desfăşurată de autor, în ultimii zece ani în cadrul Laboratoarelor IBM Canada şi USA, departamentul WebAhead Development and Security Innovations.

Teză de doctorat, Universitatea „Politehnica" Timişoara, 2009