Program PARTENERIATE - Proiecte colaborative de cercetare aplicativă - Competiţie 2013

RST - Raport ştiinţific şi tehnic extins

CALCULOS - Arhitectură cloud pentru o bibliotecă deschisă de blocuri funcţionale logice reutilizabile pentru sisteme optimizate

Codul proiectului: PN-II-PT-PCCA-2013-4-2123
Contract Nr.: 257/2014

Etapa III Algoritmi pentru control avansat şi optimizarea proceselor; arhitectura cloud

Cuprins

1 Introducere

Obiectivul principal al proiectului este proiectarea unei platforme cloud şi a serviciilor asociate, platformă care va furniza resursele de prelucrare pentru accesarea şi rularea algoritmilor de control avansat şi optimizare a instalaţiilor industriale la scară mare. Aceste servicii vor permite utilizatorului să efectueze analize de risc online şi prevenirea pericolelor folosind algoritmi generici de control, optimizare, diagnoză, prevenire a avariilor şi analiză a defecţiunilor.

Obiectivele specifice ale proiectului sunt:

- Proiectarea unei platforme care să folosească o interfaţă prietenoasă de programare adresată mai multor tipuri de utilizator;
- Crearea unor maşini virtuale care să poată găzdui module şi aplicaţii complexe;
- Reducerea costurilor de mentenanţă pentru instalaţiile industriale;
- Îmbunătăţirea relaţiei între mediul academic şi cel industrial;
- Îmbunătaţirea proceselor şi instalaţiilor industriale prin metode şi servicii accesibile.

În cadrul primei etape de execuţie a proiectului, *Etapa I — Cerinţele utilizatorului, analiza acceptabilităţii şi platforma colaborativă*, au fost efectuate de către parteneri patru activităţi principale, având ca rezultate un studiu privind stadiul industriei, cerinţele şi aşteptările sale, un raport de analiză ştiinţifică şi tehnică cu privire la posibilităţile de implementare a sistemului, elaborarea unei arhitecturi de control a proceselor şi, respectiv, analiza posibilităţilor de interfaţare cu utilizatorii şi cu procesul.

În *Etapa II — Arhitectura de sistem. Algoritmi pentru calculul riscului şi de detecţie a avariilor, diagnoză şi acomodare*, au fost prezentate elementele de bază ale unui model funcţional, specificaţiile tehnice pentru componentele sistemului, algoritmi pentru calculul riscului şi algoritmi de detecţie a avariilor, diagnoză şi acomodare.

Obiectivul etapei de execuţie. Conform planului de realizare a proiectului, în cadrul acestei etape de execuţie a proiectului, *Etapa III — Algoritmi pentru control avansat şi optimizarea proceselor; arhitectura cloud*, au fost efectuate de către parteneri următoarele activităţi principale:

- Activitatea III.1 (A3.1): Elaborare algoritmi de optimizare a sistemului,
- Activitatea III.2 (A3.2): Elaborarea unei metodologii sistematice de evaluare şi clasificare a riscurilor pentru diferite hazarduri,
- Activitatea III.3 (A3.3): Elaborare metodologie de standardizare a algoritmilor,
- Activitatea III.4 (A3.4): Implementarea algoritmilor standardizaţi.
- Activitatea III.5 (A3.5): Elaborare arhitectură cloud şi specificaţii.

Obiectivele propuse pentru această etapă a proiectului au fost atinse. Toate cele cinci activităţi prevăzute au fost efectuate şi au fost orientate spre îndeplinirea obiectivului principal şi a obiectivelor specifice menţionate mai sus.

Implementarea aplicaţiilor de conducere automată avansată a proceselor industriale necesită uzual resurse sporite de stocare şi execuţie. Raportul prezintă o aplicaţie bazată pe cloud care îi permite unui utilizator să aibă acces la diferite strategii de conducere folosind o interfaţă web, să beneficieze de regulatoare virtualizate, care să execute acei algoritmi şi să trimită unele comenzi dispozitivelor din proces. Raportul propune o soluţie pentru interconectarea modulelor implementând noi servicii, utilizând interfeţe RESTful. Abordarea inovativă prezentată presupune virtualizarea unor "baze de date" de regulatoare de proces şi îi conferă inginerului automatist dintr-o întreprindere industrială accesul la strategii avansate de modelare, optimizare şi conducere, implementate ca funcţii bloc IEC 61499.

Etapa III nu a presupus diseminarea explicită a unor rezultate. Totuşi, au fost publicate şase lucrări elaborate de unii membri ai echipei proiectului (trei dintre ele cu alţi trei colaboratori, care nu fac parte din echipă). Două lucrări au fost publicate în reviste (una cotată ISI), altă lucrare este un capitol de carte publicată de Springer, iar alte trei lucrări sunt incluse în volumele unor conferinţe internaţionale co-sponsorizate de IEEE (Institute of Electrical and Electronics Engineers) şi sunt (sau vor fi incluse) în reputata colecţie IEEE Xplore Digital Library. De asemenea, a mai fost publicată o carte cu tematică legată de cea a proiectului CALCULOS.

2 Elaborare algoritmi de optimizare a sistemului

Un domeniu de mare interes actual pe plan mondial este *fuziunea datelor distribuite* (FDD), reprezentând procesul prin care un grup de agenţi (de pildă, în contextul proiectului CALCULOS, o reţea de senzori) înregistrează date din mediul lor local, comunică cu alţi agenţi şi împreună încearcă să infereze cunoaştere despre un proces particular. Problema generală FDD include patru componente [5]: procesul, senzorii, agenţii şi comunicaţia. Măsurătorile efectuate de senzorii din proces sunt întotdeauna însoţite de erori (zgomote) şi, de regulă, măsoară doar o parte a procesului. Deşi mulţi senzori generează date simple, abilitatea de a utiliza aceste date este îngreunată de zgomote de măsură, prelucrarea preliminară a datelor (de exemplu, netezirea sau gruparea), sau de complicaţiile induse de senzorii mobili sau modificările în timp şi spaţiu ale procesului. Un agent este o entitate care poate măsura variabile din proces, estima local starea procesului, comunica informaţia şi efectua alte acţiuni (de exemplu, de deplasare). De fapt, comunicaţia are loc între nodurile unei reţele, iar un nod poate fi un agent sau un senzor pasiv.

Atunci când toate datele sunt colectate de un singur agent se obţine o *estimaţie centralizată* a procesului, care este transmisă tuturor agenţilor. Aceasta este soluţia cea mai bună, deoarece sunt utilizate toate datele. Dezantajele sunt, printre altele, transmiterea unui volum important de date în reţea şi riscul mare de eşec, prin dependenţa de un singur agent, agentul central.

Scopul FDD este de a colecta măsurătorile fiecărui senzor şi de a le prelucra local, a comunica informaţia între senzori/agenţi şi a crea estimaţii locale cât mai bune ale procesului. Cercetarea în domeniul estimării distribuite se concentrează în două zone: cea Bayesiană şi cea a consensului. Metodele Bayesiene obţin completa repartiţia statistică a procesului estimat de fiecare agent, astfel încât datele de la senzori nu trebuie memorate şi pot fi uşor fuzionate recursiv cu cunoaşterea apriorică. Metodele consensului permit agenţilor să continue difuzarea informaţiei până când obţin un acord asupra parametrilor de interes.

Lucrarea [5] tratează detaliat FDD Bayesiană, date fiind numeroasele tehnici propuse de-a lungul timpului (fuziunea Bayesiană naivă, filtrarea Kalman “federată”, filtrarea distribuită etc.). Cadrul Bayesian este teoretic foarte general şi poate fi extins riguros la numeroase clase importante de probleme de estimare distribuită, care depăşesc zona estimării liniar-Gaussiene şi filtrării Kalman. În acelaşi timp, acest cadru este practic şi oferă robusteţe estimării. Lucrarea consideră succesiv procese Gaussiene dinamice liniare, procese statice cu senzori staţionari, liniari sau neliniari, sisteme dinamice liniare sau neliniare, senzori cu dinamică, repartiţii ne-Gaussiene, comunicarea informaţiei în reţele cu topologii speciale sau generale şi sincronizarea în timp. Leitmotivul principal este filtrul informaţiei (posibil extins), care actualizează inversa matricei de covarianţă.

Un alt domeniu care trebuie luat în considerare este cel al identificării proceselor pentru obţinerea modelelor matematice utilizate în analiza sistemelor şi proiectarea regulatoarelor automate. Se are în vedere şi utilizarea expertizei existente în echipa proiectului (de pildă, [33, 27, 32, 28, 29, 31]). Chiar dacă interesul actual se îndreaptă spre identificarea sistemelor neliniare, lucru dovedit prin publicarea unui număr dedicat acestui subiect în IEEE Control Systems Magazine (vol. 36, nr. 4,

2016), tehnicile de identificare neliniară nu au ajuns încă la maturitate (spre deosebire de cele liniare). În [26] (din numărul dedicat menţionat mai sus), se consideră identificarea sistemelor liniare într-un cadru neliniar, în speţă, analiza neparametrică a distorsiunilor neliniare şi impactul acestora asupra celei mai bune aproximări liniare. Un model liniar se comportă, în general, adecvat în cazul excitării cu un zgomot cu amplitudine lent crescătoare, dar mică, dar nu când amplitudinea creşte prea mult, ceea ce poate avea loc în cazul distorsiunilor neliniare. Totuşi, identificarea sistemelor neliniare necesită mai multe date decât în cazul liniar şi, desigur, un efort de calcul mult mai mare. De aceea, este important de furnizat informaţie suplimentară pentru a garanta utilitatea investirii efortului adiţional în termenii timpului necesar, dar şi al resurselor financiare şi umane. În acest scop, în [26] se propune o procedură în trei paşi. Primul pas analizează neliniaritatea în privinţa nivelului şi a naturii acesteia, fără a mări semnificativ volumul datelor măsurate. Al doilea pas verifică dacă este suficient să se utilizeze o abordare de identificare liniară, chiar în prezenţa distorsiunilor neliniare. Se studiază proprietăţile abordării liniare în aceste condiţii şi se verifică fiabilitatea marginilor de incertitudine asociate. Ultimul pas evaluează posibilele beneficii ale utilizării unei abordări neliniare costisitoare.

Situaţia descrisă în paragraful anterior demonstrează încă o dată necesitatea de a dispune de instrumente performante pentru identificarea sistemelor liniare. Aspectele de performanţă au fost investigate în lucrările menţionate la începutul paragrafului precedent. Figura 1 [29] prezintă factorii de creştere a vitezei de calcul (rapoartele timpilor CPU) obţinute folosind funcţia `slmoen4` bazată pe algoritmul QR rapid [23, 32], în comparaţie cu funcţia MATLAB `n4sid` din System Identification Toolbox, folosind 25 seturi de date intrare-ieşire, incluzând toate cele 24 de seturi din colecţia DAISY (Database for Identification of Systems), disponibilă la adresa `www.esat.kuleuven.be/sista/daisy/`. Au fost folosite opţiunile implicite pentru `n4sid`, exceptând parametrii `order`, `'N4Weight'` and `'CovarianceMatrix'`, aleşi a fi n, `'MOESP'` şi, respectiv, `'None'`.

Algoritmii de identificare liniară au fost testaţi şi în cadrul unei configuraţii GRID şi se doreşte portarea lor în cloud. Similar, se urmăreşte şi încorporarea unor regulatoare optimale folosind algoritmi de tip Newton (recomandaţi în contextul recuperării după o funcţionare anormală) [30] (şi referinţele incluse), posibil iniţializaţi de algoritmi care exploatează structura anti-Hamiltoniană-Hamiltoniană a fascicolelor de matrice care definesc problema optimală [3].

În continuare, se prezintă câţiva algoritmi simpli de monitorizare şi optimizare.

2.1 Algoritmi optimali pentru monitorizare şi alertare industrială

2.1.1 Algoritm pe baza comportamentului normal

În realizarea unui sistem de alertare pentru situaţii de urgenţă într-o unitate de producţie se pot aborda mai multe strategii. Una dintre acestea presupune culegerea a cât mai multe date despre structura cauzală a elementelor unităţii şi combinarea acestor date cu un model de învăţare bazat pe date. Din păcate nivelul actual la care se situează algoritmii de învăţare structurală nu poate rezolva probleme cu un set masiv de variabile ascunse. O strategie alternativă pentru detecţia online a comportamentului anormal, ce nu necesită informaţii despre defectele posibile, sau un model al acestui comportament anormal, propune învăţarea unui model pentru operaţiile normale, reprezentat printr-o reţea Bayesiană. La fiecare instanţă temporală modelul este apoi folosit pentru a calcula probabilitatea setului de indicaţii ale senzorilor pentru acel pas. Apoi, prin confruntarea cu datele citite se poate evalua dacă senzorii se află în mod unitar în plaja de valori specifice operării normale. Această metodologie presupune două etape: 1) învăţarea unui model al senzorilor pentru operarea normală; 2) utilizarea modelului învăţat pentru a monitoriza sistemul, a iniţia alerte şi a realiza

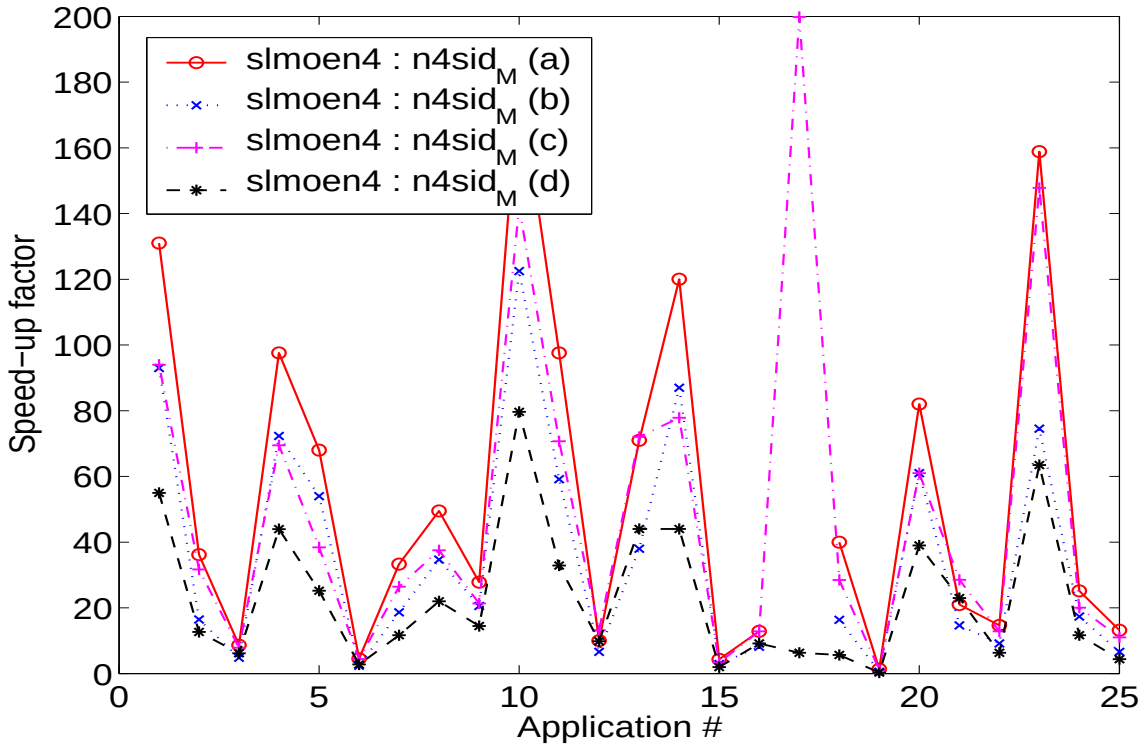

Figura 1: `slmoen4` cu algorithmul QR rapid, în comparaţie cu MATLAB 6.5.1 `n4sid` cu factorizarea QR (opţiuni implicite, cu excepţia valorilor `order` $= n$, `'N4Weight'` = `'MOESP'` şi `'CovarianceMatrix'` = `'None'`).

diagnoza online. Pe baza unui model pre-specificat al normalităţii, fiecărei componente din sistem i se asociază o stare (normal sau anormal), care este în concordanţă atât cu modelul, cât şi cu observaţiile făcute asupra sistemului.

Învăţarea unui model constă în învăţarea unei reţele bayesiene folosind o bază de date ce conţine citiri ale senzorilor reţinute în timpul operării normale a sistemului. Fiecare înregistrare a bazei de date poate fi privită ca o instanţă a procesului de producţie în ansamblu. Odată modelul învăţat, acesta poate fi confruntat cu indicaţiile senzorilor din timpul desfăşurării procesului. O cuantificare a acestei comparaţii este dată de măsura de conflict. Dacă luăm în consideraţie o probă $\bar{e} = \{e_1, e_2, \ldots, e_n\}$, unde e_1, e_2, …, e_n sunt citiri ale senzorilor, *măsura de conflict* se defineşte prin:

$$\text{conf}(\bar{e}) = \log \frac{P(e_1) \cdots P(e_n)}{P(\bar{e})}.$$

Probabilităţile $P(e_i)$ pot fi citite direct din reţeaua bayesiană în starea iniţială. Deoarece toate variabilele modelului sunt instanţiate, $P(\bar{e})$ este simplu de calculat ca fiind produsul intrărilor corespunzătoare în tabela de probabilităţi condiţionate a reţelei bayesiene. Întrucât modelul învăţat corespunde funcţionării normale a sistemului, se aşteaptă ca citirile senzorilor în timpul funcţionării normale să fie pozitiv corelate ($\text{conf}(\bar{e}) \leq 0$). Dacă însă $\text{conf}(\bar{e}) > 0$, atunci aceasta indică o funcţionare anormală şi ca atare va fi declanşată o alarmă.

2.1.2 Algoritm bazat pe clase de defect

Deoarece pentru conducerea proceselor industriale se monitorizează mii de puncte de măsură, este necesară activarea unui mecanism de selectare a caracteristicilor. Algoritmul de selecţie a caracteristicilor determină automat cei mai importanţi parametri. Toate cunoştinţele relevante sunt capturate şi integrate în baza de date astfel încât în situaţia curentă să se poată beneficia de experienţa căpătată şi însuşită (învăţată) din situaţiile anterioare. Schema unui astfel de algoritm de captură şi fuziune de cunoştinţe este prezentată în Fig. 2.

Procedura are o secţiune iniţială care are loc offline, cu intervenţie umană, şi o secvenţă automată online care rulează continuu. Partea offline este executată numai la configurarea iniţială a soluţiei. Într-o primă etapă, datele utile trebuie separate de întreaga masă de date ce sunt permanent monitorizate şi arhivate într-o instalaţie industrială. Dacă dorim să identificăm un anumit defect, trebuie luate în considerare numai măsurătorile relevante. Pornind de la documentaţia instalaţiei şi judecând în principal după criterii calitative, se face o preselecţie iniţială, separând din întreaga instalaţie doar o regiune în jurul ţintei de interes. Datele preselectate din instalaţia industrială trebuie apoi să fie preprocesate. Această procedură este comună multor aplicaţii care lucrează cu date brute sau semi-brute achiziţionate de la senzori sau traductoare. Procedura poate fi aplicată pentru mai multe operaţii, precum: transformarea unităţilor de măsură, scalare, normalizare, detectarea erorilor grosiere şi altele. Totuşi, aceste operaţii sunt relativ simple şi nu consumă foarte mult timp.

Selectarea caracteristicilor este diferită de extragerea caracteristicilor şi se efectuează prin metode specifice precum “Principal component analysis” [9], “Singular value decomposition” [14], “Manifold learning” [36] şi “Factor analysis” [37], care creează noi caracteristici extrase din cele existente, combinaţii ale celor originale. Metodele de selectare a caracteristicilor încearcă să exploreze proprietăţile intrinseci ale datelor, folosind statistica matematică sau teoria informaţiei. Selectarea caracteristicilor determină setul de parametri care sunt de interes pentru determinarea ieşirii din clasificator. Din setul mai mare de date obţinut de experţii umani în faza de preselectare este extras un subset de interes. Odată ce este definit acest subset, vor fi folosite doar aceste date din proces. Achiziţia de date în acest scop va fi restricţionată în mod corespunzător.

Validarea încrucişată este utilizată pentru antrenarea corectă a clasificatorului. Aceasta partiţionează în mod repetat setul disponibil de date în două subseturi complementare. La fiecare pas, antrenarea se face pe unul dintre ele şi testarea pe celălalt. În final, rezultatele sunt agregate, de obicei prin simpla mediere.

Se face şi o verificare suplimentară pentru a identifica eşantioane prea apropiate în ambele seturi, de antrenare şi de testare. Acestea trebuie eliminate din procedura de validare încrucişată. Odată ce este obţinut un clasificator corespunzător, acesta poate fi implementat şi folosit ca atare în faza de clasificare efectivă.

Etapa de selectare a caracteristicilor foloseşte algoritmi care îmbunătăţesc performanţa algoritmului de clasificare. Aceasta se realizează printr-o discriminare judicioasă între atributele disponibile în funcţie de particularităţile clasificatorului ce va fi folosit. Deoarece atribute corelate şi irelevante pot degrada performanţele clasificatorului, este de dorit să se aplice o metodă de selecţie a atributelor înainte de faza de antrenare. Precizia clasificatorului poate fi îmbunătăţită întroducând o fază de preselecţie corespunzător implementată înainte de faza de antrenare. În secţiunile următoare vor fi prezentanţi câţiva algoritmi performanţi ce pot fi folosiţi în etapele de învăţare, respectiv de clasificare.

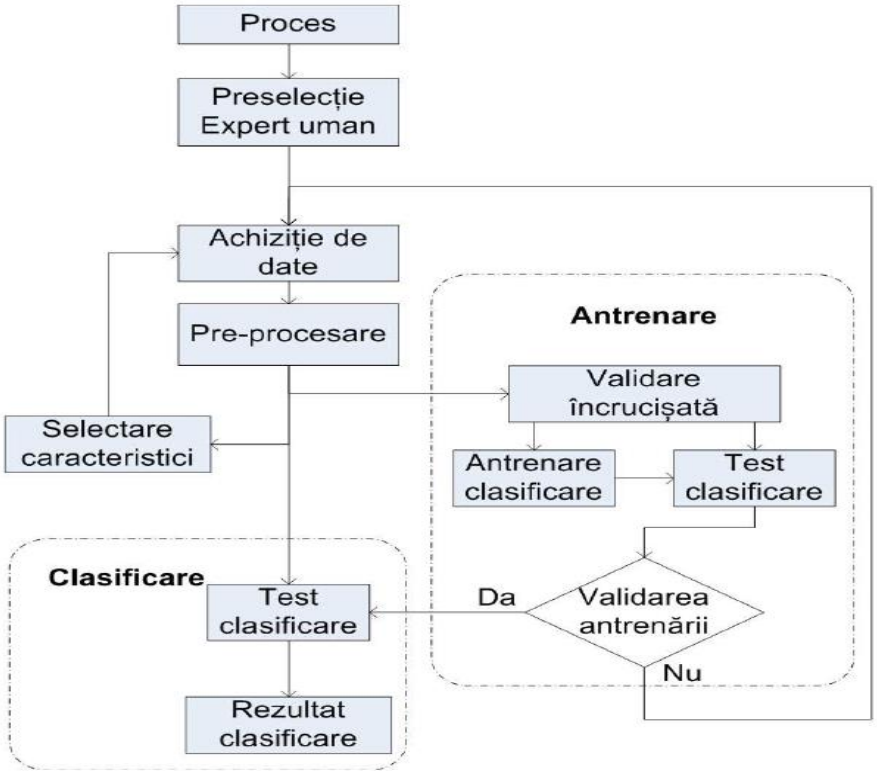

Figura 2: Algoritmul de clasificare.

2.1.3 Metode de învăţare şi de clasificare

Pentru recunoaşterea tiparelor din caracteristicile unor date reprezentate vectorial, o abordare populară este aceea de a folosi *metode nucleu* (kernel), care rezolvă problemele de eficienţă computaţională, robusteţe şi stabilitate statistică. *Eficienţa computaţională* este o caracteristică importantă a unui algoritm de învăţare automată, întrucât de obicei se pune problema unui set foarte mare de date ce trebuie analizat, iar timpul de execuţie al unui algoritm nu trebuie să fie foarte mare; *robusteţea* se referă la capacitatea algoritmului de a fi insensibil în faţa datelor perturbate de zgomot; *stabilitatea statistică* implică următorul fapt: corelaţiile găsite în caracteristici să fie într-adevăr tipare în date, adică similarităţi relevante care pot fi folosite mai departe pentru a prezice date noi în etapa de antrenare.

Pentru anumite probleme de clasificare, unele metode nucleu sunt mai potrivite decât altele. Acurateţea unei metode nucleu depinde de mai multe aspecte, precum distribuţia şi numărul de clase, zgomotul setului de date, dimensiunea setului de antrenare şi aşa mai departe. De exemplu, clasificatorul KRR (Regresia Ridge în formă duală) poate fi folosit cu succes pentru probleme în care numărul de instanţe din fiecare clasă este echilibrat, în timp ce clasificatorul KDA (Analiza discriminantă în formă duală) este mai potrivit pentru probleme cu mai multe clase. În cazuri particulare, când numărul de clase este mai mare decât 2, atunci poate apărea o problemă serioasă cu metodele de regresie.

Metoda analizei liniar discriminantă (LDA), cunoscută şi sub numele de *analiza discriminantă Fisher*, maximizează raportul dintre varianţa inter-clasă şi varianţa intra-clasă pentru a garanta separabilitatea maximă pentru un anumit set de date de antrenare.

Algoritmii de învăţare bazaţi pe metode nucleu lucrează prin scufundarea datelor într-un spaţiu

Hilbert pentru a căuta relaţii liniare în acest spaţiu folosind un algoritm de învăţare. Scufundarea se realizează implicit în momentul specificării produsului scalar între perechi de puncte, fără a fi nevoie să se furnizeze explicit coordonatele punctelor.

Funcţii nucleu. O abordare populară pentru învăţarea bazată pe similaritate este tratarea perechilor de similarităţi ca produse scalare într-un spaţiu Hilbert sau, alternativ, tratarea perechile de disimilarităţi ca distanţe într-un spaţiu euclidian. Acest lucru poate fi obţinut în două moduri. Primul mod este să se încorporeze explicit exemplele într-un spaţiu euclidian, în funcţie de similarităţile sau disimilarităţile perechilor, utilizând scări multidimensionale. Cel de-al doilea mod constă în transformarea similarităţilor în funcţii nucleu şi aplicarea metodelor nucleu. Metodele nucleu sunt definite prin două componente:

1. O funcţie ϕ care scufundă spaţiul de intrare X într-un spaţiu F (eventual de dimensiune mai mare) cu produs scalar, denumit *spaţiul caracteristicilor*.

2. Un algoritm de detectare a funcţiilor tipar liniare în spaţiul caracteristicilor F (reprezentate ca produse scalare între puncte ale spaţiului caracteristicilor).

a. *Normalizarea funcţiilor nucleu*

Un exemplu prin care se poate realiza o nouă funcţie nucleu, provenită dintr-una deja existentă, este dat de normalizarea unei funcţii nucleu existente. Dându-se un nucleu care corespunde unui vector de trăsături ϕ, normalizarea nucleului $\hat{k}(x, y)$ corespunde vectorului de trăsături dat de:

$$x \mapsto \phi(x) \mapsto \frac{\phi(x)}{\|\phi(x)\|}.$$

Normalizarea datelor ajută la îmbunătăţirea performanţelor învăţării automate pentru diferite aplicaţii practice. Cât timp setul de valori ale datelor neprelucrate poate avea variaţii mari, funcţiile de clasificare a obiectelor nu vor funcţiona optim fără normalizare. Trăsăturile sunt normalizate printr-un proces numit *standardizare*, care face ca valorile fiecărei trăsături să aibă medie zero şi varianţă unitară. Prin normalizare, fiecare trăsătură are o contribuţie aproximativ egală pentru distanţa dintre două exemple. Normalizarea funcţiilor nucleu poate fi realizată direct pe matricea nucleu.

b. *Metoda combinării funcţiilor nucleu*

Ideea de a combina funcţii nucleu vine în mod natural atunci când se doreşte îmbunătăţirea performanţelor unui clasificator. Când se combină mai multe funcţii nucleu, trăsăturile sunt scufundate într-un spaţiu dimensional mai mare. Drept consecinţă, spaţiul de căutare al unui model creşte şi clasificatorul este ajutat să selecteze o funcţie cu o valoare discriminantă mai mare. Conceptul de învăţare utilizând mai multe funcţii nucleu este cunoscut drept *multiple kernel learning* (MKL).

Cel mai natural mod de a combina două funcţii nucleu este de a le însuma. Adunarea funcţiilor nucleu sau a matricelor nucleu este echivalentă cu concatenarea vectorilor de trăsături. Dar vectorii de trăsături sunt vectori de dimensiuni mari, iar concatenarea acestora nu prezintă o soluţie viabilă dacă ne referim la spaţiu şi timp. O altă posibilitate de a obţine o combinare este de a înmulţi funcţiile nucleu. În cazul înmulţirii funcţiilor nucleu cu matrice rare, o remarcă importantă este că matricele rare vor produce o matrice nucleu mult mai rară, ceea ce nu este de dorit în anumite cazuri, deoarece pot dispărea anumite tipare.

Clasificatori liniari

Există trei mari categorii de metode de învăţare folosite în mod uzual şi anume: logica fuzzy, reţele neuronale artificiale şi "support vector machines" (SVM). Desigur, există numeroase alte metode de învăţare consacrate, o parte din acestea fiind prezentate în cele ce urmează.

Funcţiile nucleu oferă metodelor nucleu puterea de a utiliza în mod natural datele de intrare care nu sunt în format de vectori numerici, cum ar fi texte, imagini sau chiar fişiere audio sau video.

Funcţiile nucleu captează noţiunea de similaritate dintre obiecte într-un domeniu specific şi pot fi orice funcţii definite pe domeniul respectiv care sunt simetrice şi pozitiv definite. Pentru imagini, multe dintre funcţiile nucleu au aplicabilitate variată, incluzând recunoaşterea obiectelor, regăsirea imaginilor sau alte probleme similare.

În cazul problemelor de clasificare binară, algoritmii de învăţare bazaţi pe metode nucleu au nevoie de o funcţie discriminantă care adaugă $+1$ exemplelor care aparţin unei clase şi -1 exemplelor care aparţin celeilaltei clase. Această funcţie va fi o funcţie liniară în spaţiul de forma:

$$f(x) = \text{sign}\left(\langle w, \phi(x)\rangle + b\right),$$

pentru un vector de ponderi w şi o funcţie de scufundare ϕ . Nucleul poate fi utilizat atât timp cât vectorul de ponderi poate fi exprimat ca o combinaţie liniară de puncte de antrenare, implicând ca funcţia să fie definită după cum urmează:

$$f(x) = \text{sign}\left(\sum_{i=1}^{n} a_i k(x_i, x) + b\right).$$

Multe dintre metodele nucleu se diferenţiază prin modul în care găsesc vectorul w (sau, în forma duală, vectorul a echivalent).

a. *Maşini cu vectori suport (SVM)*

Clasificatorul bazat pe vectori suport (SVM) este printre cele mai utilizate metode pentru învăţarea automată şi este popular în multe probleme de recunoaştere incluzând clasificarea texturilor. SVM este conceput să maximizeze distanţa marginală dintre clase cu margini de decizie trasate utilizând diferite funcţii nucleu. SVM este proiectat să funcţioneze numai cu două clase. Acest lucru este realizat prin maximizarea separării printr-un hiperplan a celor două clase. Exemplele din vecinătate care au fost selectate pentru a determina hiperplanul sunt cunoscute drept *vectori suport*. Clasificarea multi-clase este rezolvabilă în cazul SVM, dar pentru a rezolva problema SVM pentru mai multe clase se folosesc mai mulţi clasificatori SVM binari combinaţi prin schemele de tipul "one-versus-all" sau "one-versus-one". Clasa câştigătoare este determinată de funcţia de predicţie cu valoarea cea mai mare.

b. *Regresia Ridge în formă duală (KRR)*

Regresia Ridge în formă duală (KRR) combină Regresia Ridge (regularizarea celor mai mici pătrate cu normă L_2) cu aşa numitul *kernel trick*. Acesta învaţă o funcţie liniară dată de nucleul respectiv şi de date. Pentru funcţiile nucleu neliniare, acesta corespunde unei funcţii neliniare în spaţiul original.

Regresia Ridge în formă duală selectează vectorul w care în acelaşi timp are o eroare empirică mică şi o normă mică pentru RKHS ("Reproduced Kernel Hilbert Space") generat de nucleul k. Problema de minimizare care rezultă este:

$$\min_w \frac{1}{n}\sum_{i=1}^{n}(y_i - \langle w, \phi(x_i)\rangle)^2 + \lambda\|w\|^2,$$

unde y_i este eticheta ($+1/-1$) exemplului de antrenare x_i şi λ este un parametru de regularizare.

c. *Analiza discriminantă în formă duală (KDA)*

Analiza discriminantă liniară (LDA), cunoscută şi drept *analiza discriminantă Fisher*, maximizează raportul dintre varianţa inter-clase şi varianţa intra-clase, pentru a garanta separabilitate maximală pentru un set particular de exemple.

De regulă, se apelează la abordarea LDA pentru o problemă cu două clase, sub prezumţia că cele două clase au distribuţii normale şi matrice de covarianţă identice. Faptul că se presupune că

matricele de covarianţă sunt identice implică liniaritatea clasificatorului Bayes. Astfel, LDA furnizează o proiecţie a punctelor datelor pe un subspaţiu de dimensiune 1, unde eroarea de clasificare Bayes este minimă. Metoda KDA este forma duală a algoritmului LDA, care este oarecum similar cu algoritmul KRR.

d. *Regresia de tip "Partial Least Squares" în formă duală*

Pentru problemele de regresie, se pare că uneori covarianţa vectorilor iniţiali este mai importantă decât varianţa vectorilor. Abordarea "partial least squares" (PLS) se bazează pe covarianţă pentru a dirija selectarea trăsăturilor, înainte de a face regresia celor mai mici pătrate în spaţiul de trăsături derivat. Mai precis, metoda PLS este utilizată pentru a găsi relaţiile fundamentale dintre matricea de intrare X şi matricea rezultată Y. Versiunea nucleu a PLS este un algoritm puternic care poate fi folosit şi pentru problemele de clasificare a imaginilor.

e. *Modelul celor mai apropiaţi vecini*

Algoritmul celor mai apropiaţi vecini (k-NN) este unul dintre cei mai simpli algoritmi de învăţare automată. Regula clasificatorului cel mai apropiat k vecin este descrisă în continuare.

Un obiect este atribuit celei mai apropiate clase din cele k clase din vecinătate, unde k este un număr întreg pozitiv. Dacă $k = 1$, atunci obiectul este atribuit clasei celei mai apropiate. Când $k > 1$, decizia se ia în funcţie de votul majoritar. Este convenabil să fie k impar, pentru a evita cazurile de egalitate a voturilor. Cu toate acestea, dacă avem un vot egal, obiectul poate fi atribuit celei mai apropiate clase (la distanţa minimă), sau poate să fie aleasă aleatoriu una din clasele cu vot egal pentru a fi atribuită obiectului.

Algoritmul k-NN nu este o metodă de clasificare parametrică. Astfel, nu există parametri care trebuie să fie învăţaţi. Cu alte cuvinte, modelul k-NN nu necesită deloc antrenare. Decizia de clasificare se bazează numai pe cei mai apropiaţi k vecini ai unui obiect în funcţie de o similaritate sau o distanţă. Cea mai des întâlnită măsură este distanţa euclidiană, dar pot fi utilizate şi alte măsuri. Performanţa clasificatorului k-NN depinde de numărul şi puterea discriminantă a măsurii distanţei utilizate. În faza de testare, modelul k-NN presupune calcule care cresc timpul de execuţie.

2.2 Algoritmul de optimizare PSO

2.2.1 Prezentarea algoritmului

Algoritmul Particle Swarm Optimization (PSO) este o metodă de inteligenţă artificială ce propune rezolvarea unor probleme de optimizare pornind de la analiza comportamentului social al unor grupuri, precum stolurile de păsări sau bancurile de peşti [20]. Se pare că aceste animale împărtăşesc informaţia în interiorul grupului, astfel încât comportamentul fiecărui individ este dictat de caracteristicile comportamentului întregului grup în situaţii precum căutarea hranei sau migrare. În acelaşi mod, oamenii folosesc atât experienţa personală, cât şi experienţa celor din jur, în procesul de luare a unor decizii [24]. Algoritmul PSO utilizează aceste concepte în optimizarea unor probleme cu D variabile. Studiile au arătat că este o metodă rapidă, robustă şi uşor de implementat, capabilă să găsească optimul global în probleme de optimizare continue neliniare [20, 12, 1, 24, 2]. În aplicaţii de control al proceselor, această metodă este deseori folosită pentru acordarea online sau offline a buclelor de reglare PID, pentru care s-au obţinut rezultate mai bune decât în cazul altor metode [1, 12, 25]. Fie o problemă de minimizare pentru următoarea funcţie f,

$$f(X) = f(x_1, x_2, \ldots, x_D), \tag{1}$$

unde $f : \mathbb{R}^D \to \mathbb{R}$ şi D este numărul total de variabile.

Aceasta înseamnă că trebuie să găsim X^* pentru care

$$f(X^*) \leq f(X), \quad \forall X \in S, \tag{2}$$

unde S este spaţiul de căutare.

Algoritmul PSO utilizează un grup de P particule având poziţii x_i şi viteze v_i iniţiale aleatoare într-un spaţiu D-dimensional. Fiecare particulă îşi cunoaşte cea mai bună poziţie curentă, $Pbest$, precum şi cea mai bună poziţie a grupului, $Gbest$, în spaţiul de căutare corespunzător. La fiecare pas, $Pbest$ şi $Gbest$ sunt modificate pe baza următoarelor relaţii (adaptate din [2]), pentru $i = 1, 2, \ldots, P$, $t = 1, 2, \ldots, N$,

$$Pbest_i^{t+1} = \begin{cases} Pbest_i^t, & f(X_i^{t+1}) > Pbest_i^t, \\ X_i^{t+1}, & f(X_i^{t+1}) \leq Pbest_i^t, \end{cases} \tag{3}$$

$$Gbest^t = \min_i \{Pbest_i^t\}, \tag{4}$$

unde:

P	este numărul total de particule ale grupului;
N	este numărul de iteraţii ale algoritmului;
X_i^t	este vectorul de poziţie al particulei i la momentul t;
$Pbest_i^t$	este cea mai bună poziţie a particulei i de la iniţializare şi până la momentul t;
$Gbest^t$	este cea mai bună poziţie a grupului de la iniţializare şi până la momentul t.

Viteza fiecărei particule se modifică în funcţie de propria experienţă, reprezentată de distanţa până la locaţia $Pbest$, şi, de asemenea, în funcţie de experienţa grupului, măsurată în distanţa până la locaţia $Gbest$. Modificarea poziţiei şi a vitezei fiecărei particule are loc pe baza următoarelor formule (adaptate din [2]):

$$v_{ij}^{t+1} = v_{ij}^t + c_1 r_{1j}^t \left[Pbest_{ij}^t - x_{ij}^t\right] + c_2 r_{2j}^t \left[Gbest_j^t - x_{ij}^t\right], \tag{5}$$

$$x_{ij}^{t+1} = x_{ij}^t + v_{ij}^{t+1}, \quad i = 1, 2, \ldots, P,\ j = 1, 2, \ldots, D,\ t = 1, 2, \ldots, N, \tag{6}$$

unde:

D	este dimensiunea spaţiului de căutare;
v_{ij}^t	este vectorul ce stochează viteza particulei i în dimensiunea j la momentul t;
x_{ij}^t	este vectorul ce stochează poziţia particulei i în dimensiunea j la momentul t;
$Pbest_{ij}^t$	este cea mai bună poziţie a particulei i în dimensiunea j de la iniţializare şi până la momentul t;
$Gbest^t$	este cea mai bună poziţie a tuturor particulelor din dimensiunea j de la iniţializare şi până la momentul t;
c_1 şi c_2	sunt constantele de accelerare;
r_{1j}^t şi r_{2j}^t	sunt numere aleatoare în intervalul (0, 1), generate de algoritm la momentul t.

Algoritmul se opreşte atunci când se atinge numărul maxim de iteraţii N. Soluţia problemei de optimizare este dată de:

$$X^* = Gbest = (Gbest_1, Gbest_2, \ldots, Gbest_D). \tag{7}$$

O dimensiune mare, P, a grupului înseamnă un spaţiu de căutare mai mare, ceea ce poate duce de asemenea la un număr mai mic de iteraţii necesare. Totuşi, această alegere va creşte complexitatea de calcul pe iteraţie. Studii precum [20] au arătat că de obicei se alege o valoare a lui P în intervalul [20, 60]. Numărul de iteraţii afectează de asemenea eficienţa algoritmului. De aceea, o practică recomandată este de a defini numărul maxim de iteraţii ca fiind N şi de a opri execuţia algoritmului fie când acel număr a fost atins, fie când algoritmul stagnează pentru o perioadă predefinită [24]. Constantele c_1 şi c_2 reprezintă acceleraţia cognitivă, respectiv socială, şi ele exprimă încrederea în propria poziţie, şi respectiv încrederea în grupul din care particula face parte. Aceste constante iau de obicei valoarea $c_1 = c_2 = 2$, conform studiilor existente [1, 2].

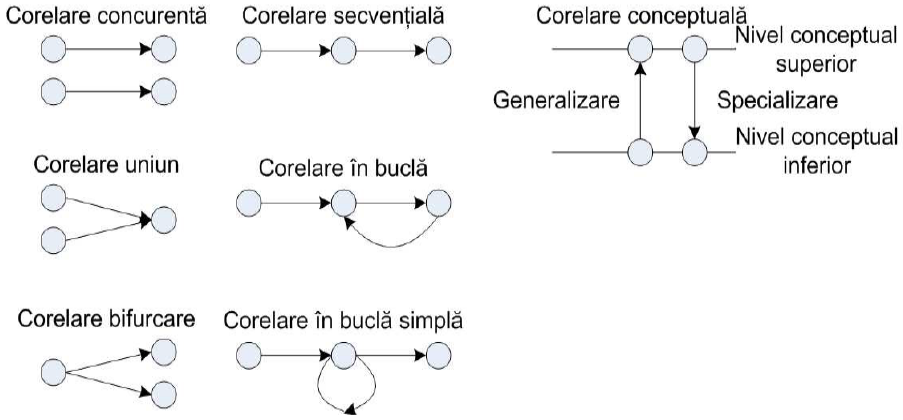

Figura 3: Interconectări între diferite procese.

3 Elaborarea unei metodologii sistematice de evaluare şi clasificare a riscurilor pentru diferite hazarduri

3.1 Model integrat de analiză a riscului folosind suport de decizie MAS

Integrarea modelelor poate fi descrisă ca o modalitate de a dezvolta modele de decizie prin adaptarea unei paradigme specifice conform cu o situaţie concretă, ceea ce conduce la realizarea unui model compozit, obţinut prin combinarea a două sau mai multe modele. Rezultă astfel un model dinamic integrat care se bazează pe un grup de rutine selectate printr-o tehnică inteligentă; acest model este denumit în cele ce urmează *model dinamic integrat pentru sisteme suport de decizie* (DSS) pentru managementul riscului industrial, asociat cu posibilitatea de producere a unor defecţiuni majore (dezastre). Pentru a optimiza acest model, se sugerează folosirea rutinelor modulare utilizate în modelul DSS ca agenţi ai unui *sistem multiagent* (MAS). În acest cadru, modelul integrat este obţinut în trei paşi:

1. selectarea unei tehnici inteligente adecvate reprezentării evenimentelor;
2. corelarea evenimentelor;
3. implementarea unei baze de cunoştinţe cu relaţii dinamice între rutine pentru un scenariu de hazard anume, cu posibilitatea dezvoltării ulterioare a acesteia.

Modelul integrat va fi prezentat în continuare ca un sistem inteligent pentru managementul dezastrelor (*Intelligent System for Disasters Management* — ISDM). Corelarea evenimentelor este una dintre tehnicile cheie în descrierea evenimentelor complexe, cu surse multiple. Sarcina corelării evenimentelor poate fi definită ca o procedură de (re)interpretare conceptuală a unui set de evenimente care au loc într-un interval predefinit de timp. Recunoaşterea unei noi situaţii printr-o procedură de corelare poate fi tratată formal ca un eveniment sintetic, putând fi totodată subiectul altor corelaţii ulterioare. Procesul de construire de noi corelaţii permite formarea unor procese complexe, interconectate. În Fig. 3 sunt descrise mai multe conexiuni fundamentale între procese, care pot fi combinate pentru a crea un mediu flexibil şi scalabil pentru modelarea situaţiilor complexe.

Utilizarea de diferite baze de date cuplate pentru monitorizarea evenimentelor de alertare a favorizat dezvoltarea şi implementarea unor sisteme integrate pentru monitorizarea informaţiilor şi

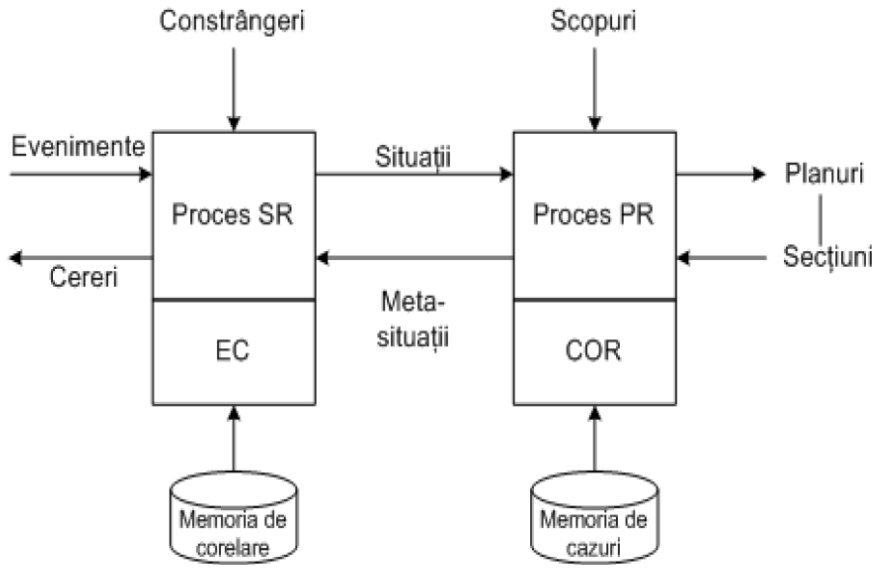

Figura 4: Relaţii reactive în procesul de luare a unei decizii.

pentru procesarea cognitivă a acestora, bazate pe management situaţional, calcul distribuit şi tehnologii multiagent. Sistemele multiagent (MAS) sunt recunoscute ca o soluţie eficientă în modelarea interacţiunii dintre un număr mare de entităţi datorită: 1) organizării structurale distribuite a MAS; 2) folosirii modelelor perceptive şi raţionale ale agenţilor mobili inteligenţi; 3) potrivirii naturale cu modelul colaborativ dintre echipele de agenţi. Aceste caracteristici ale MAS corespund cerinţelor unui ISDM, mai ales dacă în procesul decizional se recurge la arhitectura denumită *Belief-Desire-Intention* (BDI), care permite scalarea unei populaţii eterogene şi variabile de agenţi pentru mai multe sisteme de agenţi în interacţiune, cu folosirea unui tip specific de raţionare orientată pe cazuri (COR), în care fiecare caz este un şablon pentru o situaţie generică, accesibil prin accesarea unei biblioteci de cazuri şablon standard [10].

În Fig. 4 este prezentată relaţia dintre două procese principale implicate în luarea unei decizii, unul pentru *Recunoaşterea Situaţiei* (SR) prin *Corelarea Evenimentelor* (EC), care foloseşte Memoria de corelare, iar celălalt, pentru *Raţionarea pe bază de Plan* (PR) prin *Raţionarea Orientată pe Caz* (COR), care foloseşte Memoria de cazuri. Ambele procese funcţionează într-o buclă majoră în care principalele situaţii recunoscute de EC pot fi rafinate şi combinate de către COR, iar EC poate primi meta-situaţii dependente de context pentru a continua procesul de corelare a evenimentelor. În cazul informaţiilor incomplete, EC poate trimite cereri către procedurile de colectare de evenimente, solicitând informaţii adiţionale. O buclă adiţională apare în procesul PR, unde secţiuni ale unui plan pot declanşa un proces deliberativ iterativ. Principalele activităţi care pot fi asigurate de un ISDM având o arhitectură MAS de tip BDI pentru managementul riscului industrial şi al situaţiilor de urgenţă sunt:

- Aprecierea hazardului (analiza vulnerabilităţii şi a frecvenţei de producere a evenimentelor);
- Managementul riscului (analiza riscului, evaluarea şi tratarea riscului);
- Evaluarea şi îndepărtarea efectelor (elaborarea unui plan de tratare a avariilor, analiza măsurilor);
- Disponibilitate (planificarea şi managementul resurselor);

- Răspuns (proceduri de intervenţie de urgenţă, analiza şi evaluarea situaţiei curente);
- Recuperare (evaluare, probleme de relocare).

Având posibilitatea de a combina şi adapta diferite strategii, un *Sistem Inteligent pentru Managementul Dezastrelor* (ISDM) folosind un *Sistem Suport de Decizie* (DSS), care acţionează prin efortul cooperativ al mai multor agenţi inteligenţi în cadrul unei structuri multiagent, poate asigura următoarele responsabilităţi:

- Monitorizare: observarea mediului şi detectarea comportamentelor problematice;
- Generare de alarme: activarea unei alarme dacă apare o situaţie critică;
- Avertizare: avertizarea cu privire la consecinţele negative ale unei acţiuni şi sugerarea unor variante mai bune.

Pentru a genera răspunsuri pentru diferite clase de acţiuni menţionate mai sus au fost precizate patru sarcini:

- Identificarea problemei: De la analiza informaţiei primite de la infrastructura de comunicaţie sau direct de la operator, clasificatorul alege starea sistemului monitorizat;
- Diagnoză: Prezenţa evenimentelor şi situaţiilor inacceptabile necesită o explicaţie în termenii caracteristicilor de cauzalitate;
- Planificarea acţiunii: Odată ce a fost identificată o problemă, se stabileşte o posibilă succesiune de acţiuni relevante;
- Predicţie: Consecinţele evenimentelor şi ale acţiunilor operatorilor pot fi previzionate prin simulare.

O abordare generală a acestor probleme este de a concepe chiar ISDM ca un sistem multiagent, în care fiecare entitate distribuită este controlată de către un agent. Aceste sarcini locale vor fi de o complexitate mai redusă, dar totuşi interdependente. Sarcina de coordonare cu care se confruntă un astfel de sistem se referă la managementul dependenţelor între sarcinile locale, realizat de regulă printr-o metodologie orientată pe cunoştinţe. Procesul de rezolvare de probleme asociat unei sarcini este structurat în felul următor: fiecare pas poate stabili mai multe sub-sarcini, care la rândul lor trebuie rezolvate de metode mai simple şi aşa mai departe, până când se ajunge la o sarcină elementară, care poate fi îndeplinită direct. Pentru coordonarea îndeplinirii sarcinilor (înţeleasă ca managementul interdependeţelor activităţilor) trebuie să se realizeze următorii paşi:

- Detecţia dependenţelor: prin folosirea domeniului de cunoaştere cu privire la diversele dependenţe (relaţii producător-consumator, resurse limitate etc.).
- Generarea de opţiuni: pentru fiecare dependenţă este generat un set de posibile acţiuni de management.
- Decizii de management (privind acţiunile de management al dependenţei care trebuie aplicate).

3.2 Modelarea sistemelor cu informaţie incompletă folosind reţele recurente neuro-fuzzy

În lumea industrială, aproape toate procesele sunt neliniare, iar specialiştii trebuie să ţină seama de acest lucru, dacă doresc să obţină rezultate acceptabile în identificare, simulare, sau control. Mai mult, atunci când sunt disponibile doar date experimentale şi trebuie furnizat un model, metoda empirică a încercărilor succesive şi validarea acestora poate lua foarte mult timp. În acest context, abordările inteligente, precum reţelele neurale şi sistemele fuzzy, pot reprezenta soluţia căutată. Ele au capacitatea de a imita caracteristicile neliniare. În particular, sistemele fuzzy sunt cunoscute pentru capacitatea lor universală de aproximare şi îşi găsesc aplicabilitatea în probleme de modelare şi control a sistemelor complexe, incomplet definite sau cu incertitudini, acolo unde, în general, metodele standard eşuează. În cazul sistemelor dinamice neliniare, modelarea se bazează pe istoricul datelor de intrare şi de ieşire pe un interval mare de timp (dependenţă de gamă largă), de obicei colectate la momente discrete, echidistante. Numărul acestor "tacte" de întârzieri trebuie cunoscut apriori întotdeauna. În orice caz, necesitatea unei întârzieri prea mari poate mări dramatic numărul de date ale sistemului, iar aceasta conduce la creşterea exponenţială a complexităţii soluţiei.

Atunci când se doreşte aplicarea reţelelor neurale sau a sistemelor neuro-fuzzy pentru astfel de procese, ele trebuie să poată fi capabile să lucreze cu astfel de structuri. MATLAB posedă o structură şi un algoritm de învăţare — ANFIS — implementate pentru o corespondenţă directă intrare-ieşire, fără recurenţă. Şi în Simulink există un bloc care implementează ANFIS în aceeaşi manieră nerecurentă [19]. Această abordare este potrivită pentru implementarea unui regulator, care nu are nevoie decât de intrarea la momentul actual de timp pentru a furniza următoarea comandă. Pentru identificare este nevoie de mai multă informaţie. Suplimentar, dacă se vrea implementarea întârzierilor în Simulink într-o manieră recurentă folosind ANFIS, s-a constatat că buclele algebrice nu sunt permise. Soluţia propusă este de a folosi o reţea neuro-fuzzy recurentă (*Recurrent Fuzzy-Neural Network* — RFNN) cu un algoritm de învăţare adecvat structurii dinamice. Procesul dinamic care corelează nu numai intrările cu ieşirile, ci implică şi intrările anterioare, necesită o versiune modificată a algoritmului bazat pe tehnici de gradient folosit tipic în ANFIS.

Metodele de gradient propuse de mulţi autori au dezavantajul de a fi complexe, cu ecuaţii complicate şi implementări consumatoare de timp. Timpul de învăţare poate fi mai mic decât în cazul altor metode, dar efortul de scriere a codului poate fi un argument bun de a căuta o altă soluţie, fără derivate. (Există, însă, şi instrumente de generare automată a codului care calculează derivatele folosind codul sursă al funcţiilor de derivat [4].) În cele ce urmează se propune o versiune îmbunătăţită a RFNN şi se investighează fezabilitatea unui algoritm genetic diferenţial ca fiind un algoritm de învăţare optimizat.

3.2.1 Reţeaua neuro-fuzzy recurentă (RFNN)

În cadrul RFNN, trebuie folosită o buclă inversă (cu reacţie). Ieşirea creează, de obicei, dificultăţi în structura de identificare Simulink, în special în situaţia în care există mai multe bucle. În Fig. 5 este prezentată o structură fuzzy recurentă, denumită ANFIS (Adaptive Neuro Fuzzy Inference System), cu o intrare şi o ieşire.

Se notează cu $v = \{v_{ij}\}$ parametrii ce trebuie adaptaţi, unde i şi j indică locul respectivului parametru ca poziţie strat/neuron. În Fig. 6 se poate vedea structura ANFIS clasică, iar în Fig. 7 se poate urmări o diagramă generală pentru învăţarea online sau offline a RFNN. Intrarea este $u(k)$ iar ieşirea este $y(k)$. Intern, RFNN conţine o structură ANFIS cu întârzieri pentru intrări şi pentru ieşirile aduse la intrare, $y(k)$. Dacă există $m = 3$ intrări, regulile implementate sunt de forma:

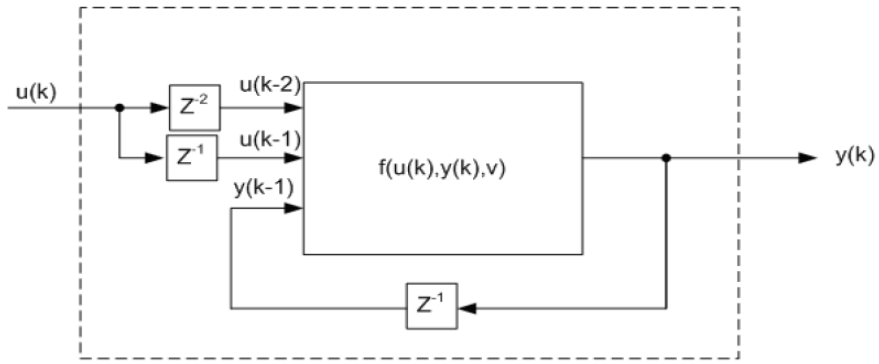

Figura 5: Reţeaua neuro-fuzzy recurentă (RFNN).

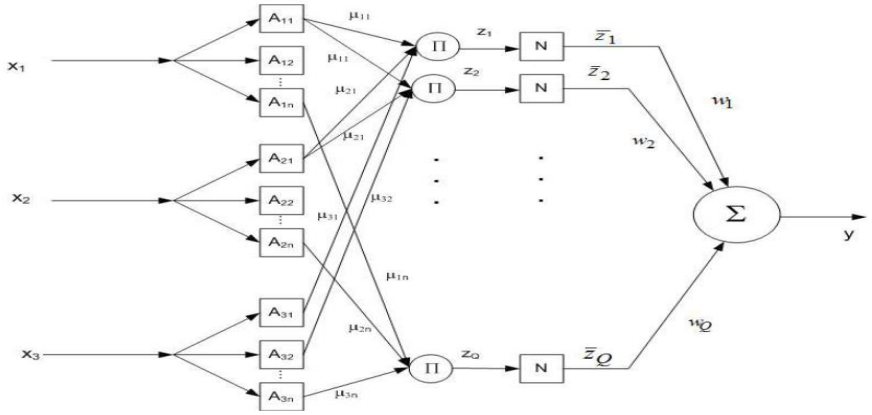

Figura 6: Schema unei reţele neuro-fuzzy recurente bazate pe structura ANFIS.

$$R_q : (x_1 = A_{1i}) \wedge (x_2 = A_{2j}) \wedge (x_3 = A_{3k}) \Rightarrow y = p_1x_1 + p_2x_2 + p_3x_3 + p_4, \quad i, j, k = 1 : n, q = 1 : Q.$$

S-a utilizat convenţia MATLAB de reprezentare a mulţimii de valori întregi i, $i + k$, ..., j ca fiind $i : k : j$. În forma de mai sus, n este numărul de funcţii de apartenenţă pentru fiecare dintre cele trei intrări. Numărul maxim de reguli este $Q_{\max} = n^m$. Fiecare funcţie de apartenenţă este definită ca o funcţie gaussiană. Pentru fiecare intrare, avem iniţial funcţii de apartenenţă distribuite uniform peste universul de discurs. Pentru a calcula ieşirea, se foloseşte procedura de propagare a semnalului feed-forward ANFIS, căreia îi corespund valorile:

$$\mu_{ij} = \mathrm{e}^{\frac{-(x_i - a_{ij})}{2b_{ij}^2}}, \quad z_q = \mu_{1i_1}\mu_{2i_2}\mu_{3i_3}, \quad \bar{z} = z_q / \sum_{i=1}^{Q} z_i,$$

$$y = \sum_{i=1}^{q} \bar{z}_i w_i, \quad w_i = p_{1i}x_1 + p_{2i}x_2 + p_{3i}x_3 + p_4, \; i = 1 : Q.$$

Înainte de a aplica procedura de identificare, RFNN trebuie antrenată. Aceasta înseamnă că reţeaua trebuie să înveţe o traiectorie dată. Tipic, această traiectorie este furnizată de date experimentale culese de la procesul neliniar care trebuie modelat.

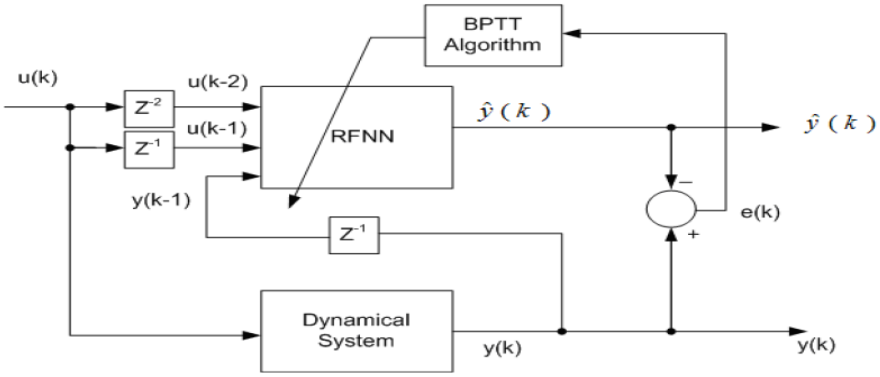

Figura 7: Diagrama de antrenare online pentru RFNN.

Se consideră un proces neliniar dat de:

$$y(t) = f(y(t-1), y(t-2), \ldots, y(t-n_y), u(t-1), u(t-2), ..., u(t-n_u)).$$

Pentru cazul particular în care avem doar trei argumente (intrări de date), modelul devine:

$$y(t) = f(y(t-1), u(t-1), u(t-2)).$$

Eroarea de ieşire instantanee, la momentul t, şi eroarea peste un orizont mai mare de timp $[t_0, t_1]$, sunt date de:

$$E(t) = \frac{1}{2\left[y(t) - \hat{y}(t)\right]^2}, \qquad E(t_0, t_1) = \sum_{t=t_0}^{t_1} \frac{1}{2\left[y(t) - \hat{y}(t)\right]^2}.$$

3.2.2 Algoritmul evolutiv diferenţial pentru optimizarea RFNN

Algoritmul evolutiv diferenţial (DE) este o metodă propusă ca o alternativă mai simplă la algoritmii genetici. DE este un optimizator de funcţii stocastice foarte simplu, bazat pe populaţii. Ideea de bază este folosirea diferenţei între doi vectori, ca metodă de perturbare a populaţiei, pentru a genera vectorii de încercare. Fiecare individ din cadrul populaţiei este reprezentat printr-un vector în care sunt codificaţi toţi parametrii ce trebuie optimizaţi pentru a minimiza eroarea. În cazul particular în care se consideră o RFNN cu $m = 3$ intrări, şi fiecare are $n = 3$ funcţii de apartenenţă gaussiene, numărul de reguli va fi RT = $(m+1)m^n = 108$, iar numărul total de vectori va fi $N_p = 2mn + (m+1)m^n = 126$.

Pentru a utiliza RFNN se aplică unele restricţii: intrările vor fi în intervalul $[-1, 1]$ sau $[0, 1]$, iar ieşirea în intervalul $[0, 1]$. De asemenea, valoarea centrală a_{ij} pentru funcţia gaussiană de apartenenţă aparţine universului de discurs şi trebuie ordonată:

$$a_{i1} < a_{i2} < \cdots < a_{im}, \quad i = 1 : m.$$

Fiecare set de parametri este codificat ca o valoare reală în intervalul $\left[U_{\text{low}}, U_{\text{high}}\right]$, iar vectorul de parametri are forma din Tab. 1.

Pe scurt, algoritmul are următorii paşi:
Pasul 0. Generarea populaţiei.
Pasul 1. Alegerea vectorului ţintă şi a vectorului sursă.

Tabela 1: Vectorul de parametri pentru optimizarea DE

μ_{11}	μ_{12}	$\dots$	μ_{ij}	$\dots$	μ_{nm}	p_{11}	p_{12}	$\dots$	p_{13}	p_{14}	$\dots$	p_{k1}	p_{k2}	p_{k3}	p_{k4}

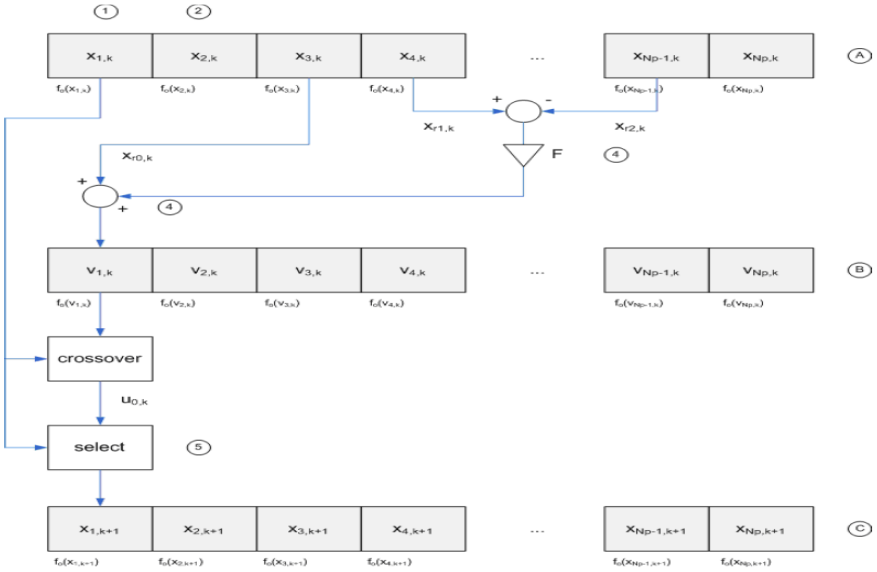

Figura 8: Algoritm evolutiv diferenţial pentru o generaţie (operaţiile între doi vectori).

Pasul 2. Alegerea aleatoare a doi membri din cadrul populaţiei (distribuţie uniformă).
Pasul 3. Calcularea vectorului ponderat de diferenţă între vectorul ţintă şi cel sursă.
Pasul 4. Adăugarea vectorului sursă.
Pasul 5. Calcularea următorului vector pe baza formulei:

$$\text{Dacă } f_0(u_{1,k}) \leq f_0(x_{1,k+1}) \text{ atunci } x_{1,k+1} = u_{1,k}, \text{altfel } x_{1,k+1} = x_{1,k}.$$

La finalul pasului 5 este generată o nouă populaţie, $P_{x,k+1}$. În Fig. 8 este prezentat sintetic algoritmul pentru generaţia k. A reprezintă populaţia iniţială, $P_{x,k}$, şi este populaţia la generaţia k, B reprezintă populaţia mutantă, $P_{m,k}$, iar C reprezintă noua populaţie, $P_{x,k+1}$.

3.3 Metode de analiză decizională multicriterială utilizate în detecţia hazardurilor

3.3.1 Consideraţii generale

Formularea problemei. Ipoteza iniţială a unei probleme de decizie multicriterială constă în existenţa a m criterii şi n variante notate C_1, C_2, …, C_m, respectiv A_1, A_2, …, A_n (ambele finite). Particularitatea acestor metode este dată de existenţa matricei decizionale, care este prezentată sub forma unei tabele (vezi Tab. 2). Fiecare rând aparţine unui criteriu şi fiecare coloană unei variante. Valoarea a_{mn} stabileşte importanţa variantei A_n funcţie de criteriul C_m. Pentru simplificare, se presupune că un scor mai bun înseamnă un loc mai înalt pe scara preferinţelor în condiţiile în care o minimizare poate fi uşor transformată într-o maximizare.

Fiecărui criteriu i se asignează o pondere notată cu w_i, care evidenţiază importanţa criteriului C_i în procesul decizional şi este reprezentată printr-un număr pozitiv. Valorile sale sunt, de regulă, determinate în urma unui proces subiectiv, fiind rezultatul evaluării unui expert sau grup de experţi.

Deşi problemele multicriteriale pot diferi, totuşi ele au o serie de caracteristici specifice:

Tabela 2: Forma generală a unei tabele decizionale

	A_1	A_2	$\dots$	A_n
C_1	a_{11}	a_{12}	$\dots$	a_{1n}
C_2	a_{21}	a_{22}	$\dots$	a_{2n}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
C_m	a_{m1}	a_{m2}	$\dots$	a_{mn}

- Existenţa criteriilor şi subcriteriilor multiple, care formează o ierarhie; orice variantă a unei probleme decizionale poate fi evaluată pe baza criteriilor care descriu proprietăţi sau caracteristici ale acestora. Unele criterii pot avea la rândul lor caracteristici care vor fi definite ca subcriterii. Pentru evaluarea unei variante, trebuie stabilit un criteriu pentru oricare proprietate.

- Existenţa criteriilor conflictuale. (Criteriile multiple se află uneori în situaţii conflictuale unele cu altele.)

- Natura hibridă este determinată de trei caracteristici: existenţa unor caracteristici imposibil de măsurat; amestecul de criterii calitative şi cantitative; existenţa criteriilor deterministe şi probabiliste.

- Gradul de incertitudine a rezultatelor (incertitudine subiectivă, informaţie incompletă).

Metode elementare de analiză decizională multicriterială. Câteva astfel de metode sunt enumerate mai jos [6].

Analiza bazată pe argumentele pro şi contra este o metodă de comparare calitativă în care lucrurile bune (pro) şi lucrurile rele (contra) sunt identificate cu privire la fiecare variantă. Listele rezultate de argumente sunt comparate una cu alta. Este aleasă varianta cu cele mai puternice argumente pro şi puţine argumente contra. Nu este nevoie de nici o pregătire matematică şi este uşor de implementat.

Metoda maximin se bazează pe o strategie care încearcă să evite cea mai slabă caracteristică prin maximizarea criteriului minim de performanţă. Varianta care are ponderea cea mai mare raportată la cel mai drastic criteriu este considerată optimă. Metoda maximin poate fi utilizată numai dacă toate criteriile sunt comparabile, astfel încât acestea pot fi măsurate pe o scală comună.

Aceste metode presupun o pondere satisfăcătoare, mai degrabă decât foarte bună, pentru fiecare criteriu. Metoda conjunctivă presupune că o variantă trebuie să respecte un prag minim de performanţă pentru toate criteriile. Metoda disjunctivă impune ca o variantă ar trebui să depăşească pragul dat pentru cel puţin un criteriu. Este eliminată orice variantă care nu respectă normele conjunctive sau disjunctive. Aceste reguli pot fi folosite pentru a selecta un subset de variante pentru analizarea cu algoritmi decizionali mai complecşi.

Criteriile în *metoda lexicografică* sunt clasificate în ordinea importanţei lor. Este aleasă varianta cu cel mai bun scor de performanţă la cel mai important criteriu. În cazul în care există mai multe variante care satisfac acest criteriu, performanţa acestora va fi comparată, până când este determinată o variantă unică.

Într-o problemă de decizie, vectorul $x = (x_1, \dots, x_n)$ joacă un rol de însumare prin includerea scorurilor de performanţă pentru fiecare criteriu şi a ponderilor asociate. Acest lucru înseamnă că vectorul x trebuie să se încadreze cât mai bine în liniile matricei deciziei.

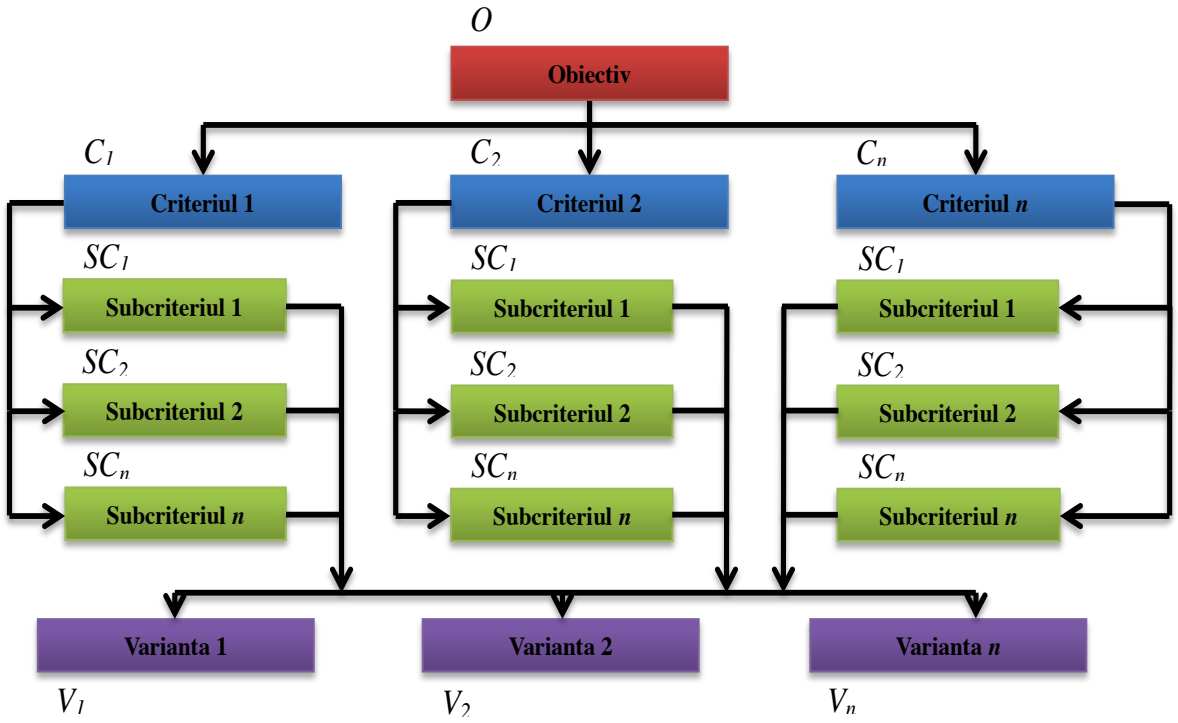

Figura 9: Reprezentarea structural-ierarhizată a unei probleme decizionale multicriteriale.

Modelul general. Metoda de analiză multicriterială poate să crească în complexitate prin rafinarea criteriilor, introducerea de subcriterii şi optimizarea procesului de alocare a ponderilor. În Fig. 9 poate fi observată reprezentarea grafică structural-ierarhizată utilizată în cazul unei probleme decizionale multicriteriale, în care notaţiile sunt următoarele: O este obiectivul vizat, $C_1, \ldots, C_n$, respectiv $\mathrm{SC}_1, \ldots, \mathrm{SC}_n$, sunt criteriile şi subcriteriile necesare atingerii acestuia şi $V_1, \ldots, V_n$, sunt variantele considerate potrivite pentru satisfacerea criteriilor şi subcriteriilor.

3.3.2 Metoda Analitică Ierarhizată

Definiţii. *Metoda Analitică Ierarhizată* (MAI) este o abordare de luare a deciziilor multicriteriale care a atras interesul multor cercetători, în principal datorită simplităţii algoritmului matematic şi a faptului că datele de intrare necesare sunt destul de uşor de obţinut. Ca instrument de suport decizional, este folosită pentru a rezolva probleme complexe prin intermediul unei structuri ierarhice multi-nivel a obiectivului, criteriilor, subcriteriilor, şi variantelor. Datele de intrare sunt obţinute prin utilizarea unui set de comparaţii perechi care sunt folosite apoi pentru a obţine ponderile criteriilor de decizie, precum şi pentru măsurarea performanţei relative a variantelor în funcţie de fiecare criteriu. În cazul în care comparaţiile nu sunt perfect determinate, atunci se poate aplica un procedeu matematic pentru îmbunătăţirea acestora.

Considerând mulţimile de criterii $C = \{C_1, C_2, \ldots, C_n\}$, subcriterii $\mathrm{SC} = \{\mathrm{SC}_1, \mathrm{SC}_2, \ldots, \mathrm{SC}_n\}$, variante $V = \{V_1, V_2, \ldots, V_n\}$, unde $n > 1$, dar finit, şi notând cu a_{ij} cuantificările obţinute în urma aplicării Tabelei 2, debutul algoritmului MAI este de constituire a matricelor decizionale

Tabela 3: Forma tabelară a matricei decizionale A

	V_1	V_2	$\cdots$	V_n
V_1	a_{11}	a_{12}	$\cdots$	a_{1n}
V_2	a_{21}	a_{22}	$\cdots$	a_{2n}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
V_n	a_{n1}	a_{n2}	$\cdots$	a_{nn}

Tabela 4: Forma tabelară a matricei decizionale A cu diagonala principală unitară

	V_1	V_2	$\cdots$	V_n
V_1	1	a_{12}	$\cdots$	a_{1n}
V_2	a_{21}	1	$\cdots$	a_{2n}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
V_n	a_{n1}	a_{n2}	$\cdots$	1

comparative cu forma generală:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \tag{8}$$

În multe lucrări de cercetare s-a adoptat o formă tabelară a matricei A, în vederea prelucrării ulterioare mai uşoare a datelor (Tab. 3, cu $m = n$). Analizând diagonala principală a matricei decizionale ale cărei elemente componente sunt de forma a_{ii} se poate interpreta că acestea sunt rezultatul cuantificării între două variante identice (V_1 şi V_1, V_2 şi V_2, …, V_n şi V_n). Ca urmare a aceastui fapt, se poate enunţa următoarea definiţie:

Definiţia 1: În cazul comparării unei variante V_i cu ea însăşi, cuantificarea va avea valoarea 1.

Matricea A devine cea din Tab. 4. Luând în considerare valoarea a_{ij} corespunzătoare cuantificării a două variante, reciprocitatea cuantificării poate fi enunţată de asemenea astfel:

Definiţia 2: Considerându-se cunoscută cuantificarea a_{ij} a două variante V_i şi V_j, $i \neq j$, reciproca acesteia va fi exprimată de relaţia: $a_{ij} = 1/a_{ji}$.

Forma finală a matricei A este redată în Tab. 5. Avându-se în vedere faptul că datele de intrare folosite în constituirea matricei A sunt obţinute ca urmare a cuantificării între criterii, subcriterii şi variante cu ajutorul factorului uman, se ia în calcul şi o anumită doză de subiectivitate specifică acestuia, denumită *grad de eroare*.

Considerând cunoscute datele primare de intrare definite de mulţimile O, C, SC, precum şi cuantificările a_{ij} asociate acestora, pasul următor al algoritmului constă în normalizarea matricei decizionale A. Dată fiind diversitatea scalelor pe care pot fi măsurate variantele, este necesar ca punctajele diferitelor criterii să fie transformate, putând fi comparate numai dacă referinţele sunt aceleaşi. Această transformare este numită *standardizare* sau *normalizare*. Unităţile de măsură sunt uniformizate printr-o funcţie valorică sau procedură de standardizare, iar punctajele acestora îşi pierd dimensiunea şi unitatea de măsură aferentă.

MAI nu menţine ordinea de mărime relativă, ci stabileşte punctajele clare ale opţiunilor în intervalul [0,1]. Pentru aceasta, se aplică normalizarea brută, prin utilizarea formulei (10) în care a_{ij}^*

Tabela 5: Forma finală a matricei decizionale A

	V_1	V_2	$\cdots$	V_n
V_1	1	a_{12}	$\cdots$	a_{1n}
V_2	$1/a_{12}$	1	$\cdots$	a_{2n}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
V_n	$1/a_{1n}$	$1/a_{2n}$	$\cdots$	1

Tabela 6: Forma normalizată a matricei decizionale A

a^*_{11}	a^*_{12}	$\cdots$	a^*_{1n}
a^*_{21}	a^*_{22}	$\cdots$	a^*_{2n}
$\vdots$	$\vdots$	$\vdots$	$\ddots$
a^*_{m1}	a^*_{m2}	$\cdots$	a^*_{mn}

reprezintă valoarea normalizată a cuantificării a_{ij}, iar rezultatele au forma din Tab. 6.

$$a^*_{ij} = a_{ij} / \sum_{k=1}^{m} \sum_{l=1}^{n} a_{kl}. \tag{9}$$

Finalizarea acestor calcule declaşează cel de-al doilea pas în algoritmul MAI, care are ca scop obţinearea ponderilor parţiale, care sunt exprimate, de regulă, sub formă de procente. Necesitatea acestei etape este foarte clară, deoarece furnizează o clasificare intermediară a variantelor funcţie de criterii şi subcriterii. Formula care stă la baza acestor rezultate este dată de (10), unde w_n reprezintă gradul de preferinţă (ponderea alternativei în preferinţele decidenţilor) şi n rangul matricei decizionale:

$$w_n = \frac{1}{n} \sum_{k=1}^{m} a^*_{kn}. \tag{10}$$

Ultimul pas al MAI este realizarea matricei de putere care va furniza clasamentul final al variantelor, şi anume, funcţie de obiectivul vizat. Considerând $w_{C_1}, \ldots, w_{C_n}$, $w_{V_1}, \ldots, w_{V_n}$, ponderile parţiale ale mulţimilor C şi V, forma generală este dată în Tab. 7: Formula de calcul a clasamentului final este (11), iar rezultatele alcătuiesc vectorul W al preferinţelor finale:

$$W_{vn} = \left(\sum w_{C_n} w_{V_n} \right) \cdot 100\%. \tag{11}$$

Pentru $m = n$, determinarea vectorului W corespunzător valorii proprii $\lambda_{\max}$ presupune rezolvarea ecuaţiei caracteristice a matricei A. Funcţia caracteristică a acesteia este dată în (12), iar ecuaţia

Tabela 7: Forma matricei de putere

	Criterii		
	w_{C_1}	w_{C_2}	w_{C_3}
Variante	w_{V_1}	w_{V_1}	w_{V_1}
	w_{V_2}	w_{V_2}	w_{V_2}
	w_{V_3}	w_{V_3}	w_{V_3}

caracteristică are forma polinomială din (13),

$$f(\lambda) = |A - \lambda I_n| = \begin{vmatrix} a_{11} - \lambda & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} - \lambda & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} - \lambda \end{vmatrix}, \tag{12}$$

$$f(\lambda) = |A - \lambda I_n| = 0 \Rightarrow c_0\lambda^n + c_1\lambda^{n-1} + \cdots + c_{n-1}\lambda + c_n = 0. \tag{13}$$

Vectorii proprii ai matricei A satisfac $(A - \lambda I_n)x = 0$, cu $x \neq 0$. Pentru λ_n, este valabilă ecuaţia (14) cu vectorul $x_n \neq 0$,

$$(A - \lambda_n I_n)x_n = 0. \tag{14}$$

Se presupune că $x_n = W$ în cazul valorii $\lambda_{\max}$, iar ecuaţia (14) devine:

$$AW = \lambda_{\max} W. \tag{15}$$

În continuare, este prezentată succint o metodă de identificare a clasamentului final.

Metoda puterilor permite calculul vectorului propriu al variantelor cu grad de precizie variabil funcţie de iteraţiile succesive de aproximare ale acestuia. Astfel se stabileşte valoarea dorită a preciziei ε ce caracterizează vectorul W. Paşii iterativi vor fi notaţi cu k, $k = 1, 2, \ldots, m$, iar numărul iteraţiilor va fi m. Fiecare pas va obţine o aproximare îmbunătăţită, w_k, a vectorului propriu al lui A corespunzător lui $\lambda_{\max}$. Dacă la un moment dat este atinsă precizia ε, acea aproximare a vectorului propriu corespunzător lui $\lambda_{\max}$ este considerată ca fiind vectorul W. Prima aproximare a vectorului propriu are forma:

$$W^1 = \left[n^{-1}, \ldots, n^{-1}, \ldots, n^{-1}\right]^T. \tag{16}$$

În cazul aproximărilor succesive ($k \geq 2$), se introduce vectorul $d^k = AW^{k-1}$. Aproximările succesive ale vectorului W^k sunt

$$W^k = d^k / \|d^k\|. \tag{17}$$

Sfârşitul calculelor este realizat când aproximările devin mai mici ca ε:

$$\|W^k - W^{k-1}\| < \varepsilon. \tag{18}$$

După cum s-a menţionat în debutul acestui subcapitol, MAI compară variantele între ele funcţie de criteriile şi subcriteriile convenite. Rezultatele acestor acţiuni sunt seturi de matrice şi clasamentul final.

Evaluarea corectitudinii rezultatelor obţinute prin MAI. Matricea decizională A este considerată perfect determinată (erorile datorate subiectivităţii şi greşelii în interpretare sunt nule) atunci când, pentru criteriile i şi j, este valabilă relaţia $a_{ij} = a_{ik}a_{kj}$. În cazul în care elementele comparate ar fi exprimate în aceleaşi unităţi de măsură, procedeul de comparare nu ar mai fi necesar, fiind de ajuns doar determinarea valorii maxime a acestor unităţi. Presupunând cuantificările a_{ij} ca fiind rezultatul ponderilor variantelor i, respectiv j, notate cu p_i şi p_j, a_{ij} este definit de relaţia $a_{ij} = p_i/p_j$. Simetria comparaţiei este valabilă şi în acest caz, astfel că $a_{ji} = 1/a_{ji} = p_j/p_i$. Aplicând formula de mai sus pentru a_{ij} se obţine

$$\begin{bmatrix} p_1/p_1 & p_1/p_2 & \cdots & p_1/p_n \\ p_2/p_1 & p_2/p_2 & \cdots & p_2/p_n \\ \vdots & \vdots & \cdots & \vdots \\ p_n/p_1 & p_n/p_2 & \cdots & p_n/p_n \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{bmatrix} = n \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{bmatrix}. \tag{19}$$

Tabela 8: Valorile RI funcţie de rangul matricei comparative A

n	2	3	4	5	6	7	8
RI	0	0,58	0,9	1,12	1,24	1,32	1,41

Înlocuind forma matematică a matricei cu notaţia A, rezultă că $AW = nW$, şi egalând cu (15) se obţine $\lambda_{\max} = n$. Dacă matricea A este perfect determinată, ca în cazul de mai sus, atunci fiecare linie este înmulţită cu constanta sa specifică. În acest caz toate valorile proprii λ_i, cu excepţia uneia, sunt egale cu zero, dar $\sum \lambda_i = \mathrm{tr}(A)$, unde tr($A$) este "urma" matricei A, calculată prin însumarea elementelor diagonalei principale.

Având în vedere cele enunţate până acum, se poate da următoarea definiţie:

Definiţia 3: Matricea A este perfect determinată dacă şi numai dacă $\lambda_{\max} = n$.

În cazul în care $\lambda_{\max} > n$, A conţine o anumită eroare, deci clasamentul final nu este exact.

În continuarea analizei, vom considera că p_i şi p_j sunt măsurate exact, adică, $a_{ij}\frac{p_j}{p_i} = 1$, şi coincid cu răspunsurile experţilor, adică, $\sum_{j=1}^{n} a_{ij} \cdot \frac{p_j}{p_i} = n$. În cele mai multe cazuri relaţiile de mai sus nu coincid cu realitatea. Abaterile de la măsurătorile precise sunt compensate de introducerea unui indice $\delta_{ij} > -1$, deci, $a_{ij} = (1 + \delta_{ij})\frac{p_i}{p_j}$. Utilizând ecuaţia (15) se deduce că

$$\lambda_{\max} - n = \frac{1}{n} \sum_{i=1}^{n} \frac{\delta_{ij}^2}{1 + \delta_{ij}} \geq 0. \tag{20}$$

Numărul obţinut în urma diferenţei din membrul stâng al ecuaţiei este expresia abaterii matricei studiate de la cea perfect determinată. Este nevoie, aşadar, de introducerea unor indicatori care să determine dimensiunea acestei abateri (deci a corectitudinii determinării experţilor), conform formulei:

$$\mathrm{CI} = \frac{\lambda_{\max} - n}{n - 1}. \tag{21}$$

Acest indice al abaterii se referă la media soluţiilor disponibile pentru ecuaţia caracteristică a matricei studiate.

În continuare, trebuie calculată abaterea decizională a experţilor, care se realizează după formula $\mathrm{CR} = \frac{\mathrm{CI}}{\mathrm{RI}}$. Factorul RI se obţine din tabele (Tab. 8). În urma parcurgerii acestor paşi, procentul final al erorii nu trebuie să depăşească valoarea de 10%. În caz contrar, raţionamentul trebuie refăcut complet, deoarece s-a produs o eroare majoră la nivelul determinării numărului criteriilor, sau pe parcursul cuantificărilor propuse de experţi.

În concluzie, metoda Analitică Ierahizată are la bază un aparat matematic suficient de simplu încât să poată fi aplicat pe scară largă în multiple domenii tehnico-economice, dar are unele limitări date de aproximările succesive apărute în timpul calculelor şi de subiectivismul factorului uman; de aceea, atunci când este utilizată în analiza avariilor, trebuie să se studieze cu mare atenţie evoluţia istorică a procesului pentru stabilirea cât mai corectă a ponderilor acordate criteriilor. Pe de altă parte, nu trebuie neglijat câştigul de timp şi resurse realizat prin aplicarea algoritmilor matematici de rezolvare a problemelor decizionale şi faptul că pot fi instrumente performante în ghidarea către varianta optimă.

4 Elaborare metodologie de standardizare a algoritmilor

4.1 Principii de urmat în dezvoltarea algoritmilor

Biblioteca trebuie să asigure că algoritmii puşi la dispoziţia utilizatorului respectă anumite principii de funcţionare, astfel încât să nu afecteze în mod negativ stabilitatea, fiabilitatea şi/sau eficienţa aplicaţiilor în care sunt integraţi. Algoritmii bibliotecii trebuie să se conformeze unor standarde specifice de documentare şi implementare, aşa încât să se asigure o interfaţă uniformă cu utilizatorul, mentenanţă uşoară a înregistrărilor şi adaptabilitatea şi portabilitatea necesare pentru execuţia pe diferite platforme. Pentru a putea fi adăugaţi în bibliotecă, algoritmii vor parcurge întâi un proces de evaluare, care să confirme îndeplinirea criteriilor de calitate şi de reutilizabilitate enunţate în continuare. Biblioteca nu va trebui să conţină un număr prea mare de înregistrări şi se vor elimina duplicatele. Pentru implementarea unor funcţionalităţi similare, se poate deseori folosi o singură componentă având unul sau mai multe moduri de funcţionare ce pot fi selectate prin intermediul unor parametri externi.

4.1.1 Asigurarea calităţii algoritmilor

Principalele criterii pe care trebuie să le îndeplinească un algoritm pentru a putea fi adăugat în bibliotecă sunt:

- *Utilitatea*: algoritmul trebuie să rezolve o problemă care are o necesitate practică;
- *Robusteţea*: algoritmul trebuie să ofere rezultate corecte sau un mesaj de eroare în cazul în care este utilizat incorect (de exemplu, în cazul în care problema nu este bine condiţionată sau nu este inclusă în clasa de probleme pentru care a fost proiectat algoritmul);
- *Stabilitate numerică* şi *precizie*: să ofere rezultatele aşteptate, similare celor obţinute prin evaluarea numerică a reprezentării matematice;
- *Viteza de execuţie*: să fie cât de mare cu putinţă, fără însă a afecta robusteţea, stabilitatea numerică şi precizia;
- *Unicitatea*: algoritmul nu trebuie să se mai regăsească în bibliotecă.

În scopul asigurării acestor criterii, fiecare algoritm va avea asociat un fişier de descriere care să detalieze utilitatea şi aplicabilitatea algoritmului, domeniul de utilizare, formulele matematice care au stat la baza reprezentării, precum şi rezultatele aşteptate. Înainte de a fi adăugaţi în bibliotecă, algoritmii vor fi testaţi într-un mediu simulat, iar rezultatele vor fi disponibile în aplicaţia web.

4.1.2 Asigurarea reutilizabilităţii algoritmilor

Pentru asigurarea reutilizabilităţii algoritmilor trebuie definită o strategie şi un set de reguli a reprezentării acestora. Aceste reguli asigură criteriile de performanţă ce trebuie respectate de componentele bibliotecii astfel încât să se poată îndeplini obiectivul principal, acela de a pune la dispoziţia utilizatorilor o bibliotecă deschisă de algoritmi reutilizabili ce pot fi folosiţi pentru controlul proceselor industriale. În [35] este propusă o metodă de proiectare a unor regulatoare bazate pe funcţii bloc reutilizabile, independente de aplicaţia concretă în care vor fi folosite, care să poată gestiona toate situaţiile ce pot apărea. Metoda propusă este exemplificată pe proiectarea unui regulator pentru o componentă mecatronică, însă ea poate fi adaptată şi pentru elementele bibliotecii de algoritmi reutilizabili, menţionate ca funcţii bloc reutilizabile în cele ce urmează. Această metodă presupune

parcurgerea unor paşi necesari proiectării unui regulator independent de aplicaţie, la care se adaugă componentele specifice rezolvării unei anumite probleme, iar în final, componentele de interfaţare cu restul sistemului. Paşii care definesc comportamentul funcţiei bloc independent de aplicaţie sunt:

- *Specificarea funcţionalităţii*. Funcţionalitatea se referă la operaţiile ce trebuie efectuate prin intermediul algoritmilor definiţi în cadrul funcţiei bloc. Este important de notat că echipamente similare pot avea funcţionalităţi diferite, ceea ce impune şi funcţii bloc de control diferite;

- *Autopercepţia*. Funcţia bloc trebuie să analizeze informaţiile pe care le primeşte ca date de intrare de la senzori. Astfel funcţia bloc are o percepţie asupra contextului în care se execută, aşa încât să îşi adapteze starea la acel context. Altfel spus, toate datele de intrare trebuie asociate unui eveniment de intrare pentru ca ele să îşi actualizeze valoarea în momentul modificării mărimilor primite de la senzori;

- *Iniţializarea*. Iniţializarea unei funcţii bloc are rolul atribuirii unor valori variabilelor de ieşire înaintea execuţiei normale a acestei funcţii. Aceasta se face prin intermediul unui eveniment de intrare INIT care declanşează execuţia secvenţei de iniţializare.

Aceşti paşi sunt cei care asigură definirea funcţiei bloc în interiorul unui proces. Chiar dacă au fost definiţi algoritmii care implementează funcţionalitatea, la acest nivel nu se ştie în ce condiţii aceştia vor fi executaţi şi nici impactul pe care execuţia lor îl are asupra elementelor din proces. Pentru adaptarea unei funcţii bloc la o problemă specifică trebuie definite următoarele caracteristici:

- *Percepţia mediului*. Aceasta însemnă că funcţionarea poate fi condiţionată de anumite limite specifice procesului (de exemplu, domenii de variaţie, în cazul proceselor continue, sau informaţii referitoare la limite fizice, care să permită, de pildă, unui braţ al unui robot să evite coliziunea cu alte obiecte);

- *Percepţia sarcinilor*. Aceasta presupune ca funcţia bloc să primească anumiţi parametri de intrare care să îi permită să înţeleagă ce se doreşte să se obţină. Aceşti parametri pot fi primiţi de la alte blocuri sau de la senzorii din proces şi pot fi, de asemenea, transmişi mai departe către alte funcţii bloc. Acest concept este reprezentat prin condiţiile tranziţiilor dintr-o diagramă de execuţie ECC;

- *Decizia*. În urma analizei datelor de intrare, funcţia bloc trebuie să decidă acţiunile ce urmează a fi întreprinse. Aceste decizii sunt reprezentate de tranziţiile ilustrate într-un ECC. Tranziţiile sunt fie condiţionate de valoarea de adevăr a expresiilor pe care le conţin, fie se execută instantaneu în cazul în care condiţia este simbolizată cu 1.

Ultimul aspect influenţează proiectarea unei funcţii bloc în scopul integrării într-un sistem distribuit. Acesta se referă la modul în care funcţia interacţionează cu restul componentelor sistemului. Între componentele unui sistem există un schimb continuu de evenimente şi date astfel încât să se realizeze un scop comun. Acest schimb se poate clasifica în informaţii (care sunt date ce trebuie prelucrate şi care influenţează procesul de decizie) şi comenzi (care trebuie executate, fără a fi necesară procesarea în cadrul funcţiei). Întrucât poate fi dificilă identificarea numărului de evenimente de intrare şi de ieşire necesar interfeţei unei funcţii bloc, astfel încât să fie îndeplinite toate cerinţele, în [35] se propune o uniformizare a numelui şi tipului lor, ceea ce face să fie posibilă asocierea cu uşurinţă între acestea şi datele de intrare. Se recomandă clasificarea evenimentelor şi a datelor după cum urmează.

- *Iniţializare*. Evenimentele de iniţializare INIT-INITO există definite în majoritatea funcţiilor bloc şi trebuie conectate numai la acele date care sunt implicate în acest proces.
- *Percepţie*. Aceste tipuri de evenimente (propuse ca fiind perechea REQ-CNF) sunt responsabile pentru schimbul de informaţii cu alte blocuri sau cu elementele din proces (senzori şi elemente de execuţie). Acestea trebuie utilizate numai pentru actualizarea datelor de intrare şi nu vor fi utilizate pentru condiţionarea tranziţiilor din diagrama ECC. În cazul aplicaţiilor de mari dimensiuni se pot defini mai multe astfel de perechi care să aibă asociate grupuri de date de intrare, eficientizând astfel execuţia.
- *Ordonarea evenimentelor*. Se propune perechea de evenimente IN_FBD-OUT_CMD care vor fi folosite atunci când o funcţie bloc comandă altă funcţie bloc, de la care primeşte un feedback. Pentru fiecare comandă independentă (folosită într-o singură operaţie) trebuie adăugate în ECC o stare şi o tranziţie. În cazul în care pentru efectuarea unei tranziţii este necesară o combinaţie de comenzi, atunci evenimentul OUT_CMD va fi însoţit de variabilă booleană care va condiţiona tranziţia. Evenimentul IN_FBD are rolul de a indica disponibilitatea funcţiei bloc ce este comandată pentru a executa o nouă comandă.
- *Execuţia evenimentelor*. Presupune perechea de evenimente IN_CMD-OUT_FBD, aplicată unei funcţii bloc care primeşte comezi de la altă funcţie bloc, pe care le execută pentru ca apoi să genereze evenimentul de feedback.

4.1.3 Structura algoritmilor

Algoritmii vor fi reprezentaţi sub forma unor funcţii bloc compuse, cu o structură deschisă, astfel încât utilizatorul să poată efectua modificări în cazul în care doreşte acest lucru. Vor putea fi executaţi pe platforma cloud numai algoritmii disponibili în platforma cloud, care au fost, în prealabil, testaţi şi validaţi de un administrator de sistem, nu şi algoritmii modificaţi de către un utilizator. Algoritmii vor fi disponibili atât sub formă editabilă, cu specificarea limbajului de programare utilizat, cât şi în formă executabilă. Forma editabilă va putea fi accesată de către utilizator în scopul adaptării la un alt limbaj de programare sau al adaptării la o aplicaţie specifică.

Structura algoritmilor editabili, elaboraţi în IEC 61499. Algoritmii vor putea fi disponibili sub forma unor fişiere de tip *.fbt compatibile cu toate mediile de dezvoltare care implemntează standardul IEC 61499. Fiecare algoritm va fi însoţit de o scurtă descriere care să detalieze funcţionalitatea îndeplinită de acesta, dacă este aplicabil numai unor anumite procese, domeniul de utilizare, denumirea şi rolul parametrilor de intrare şi de ieşire (date şi evenimente) şi eventual rezultatul aşteptat în urma implementării. Algoritmii care implementează funcţii mai complexe (de modelare, optimizare, analiză etc.) vor putea fi executaţi în bibliotecă pentru ca apoi rezultatul să fie trimis către un echipament aflat la distanţă. Pentru ca acest lucru să fie posibil, funcţia bloc compusă specifică algoritmului va fi introdusă într-o structură de tip configuraţie de sistem cu extensia *.sys având definit un dispozitiv şi o resursă. Aceasta este configuraţia minimă care permite simularea unui sistem distribuit şi oferă în plus facilităţi de operare specifice echipamentelor industriale precum pornire la cald, pornire la rece sau oprire. Exemplificarea unei configuraţii de sistem pornind de la algoritmul PSO (Particle Swarm Optimization) este ilustrată în Fig 10. Blocurile ce trebuie obligatoriu adăugate sunt cele din chenarul de sus care asigură controlul execuţiei (prin blocurile RUNSTOP şi RS_GATE) şi simulează ceasul unui echipament (prin blocul de tip E_CYCLE) pornind de la o perioadă de eşantionare stabilită (definită în blocul DT), precum şi cele care asigură interfaţarea cu aplicaţia web şi cu procesul. În acest exemplu au fost adăugate suplimentar blocuri de preluare a

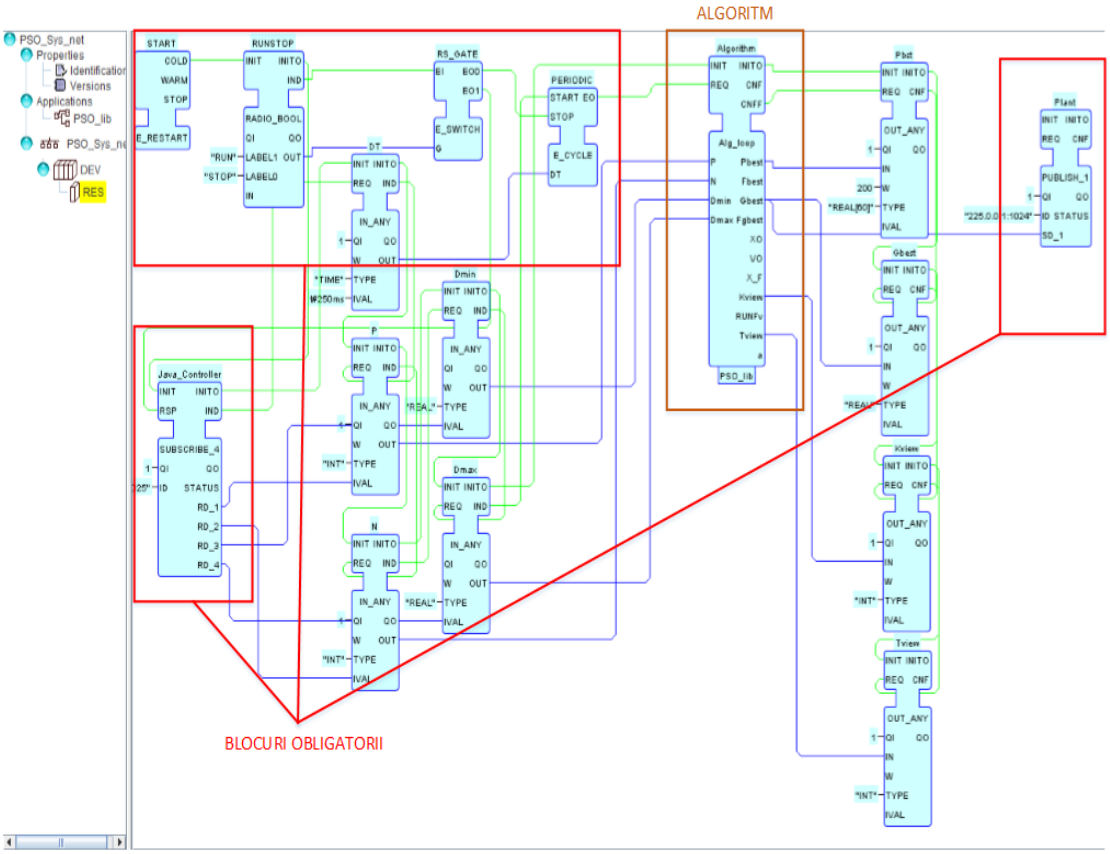

Figura 10: Exemplu de construire a unei configuraţii de sistem.

parametrilor de intrare şi de ieşire din algoritm care permit urmărirea mai uşoară a execuţiei algoritmilor şi a recepţiei corecte a parametrilor de la pagina web din cadrul serverului de aplicaţie. Pentru integrarea în sistemul de execuţie online al bibliotecii, funcţiei bloc compuse i se va adăuga un bloc Read_data conectat la parametrii de intrare ai funcţiei bloc, care va permite preluarea parametrilor de execuţie de la interfaţa web, şi un bloc de Send_data la parametrii de ieşire ai funcţiei bloc, care va asigura transmiterea rezultatului către echipamentul aflat la distanţă în instalaţie. Sunt cazuri în care algoritmii se execută în funcţie de unul sau mai mulţi parametri de proces. Pentru aceasta este necesară adăugarea unui bloc de Read_data suplimentar de conectare la acei parametri.

Pentru ca procesul de generare a IP-urilor de la interfaţa web să se poată face automat, blocurile de comunicaţie vor avea parametrul ID sub forma unui identificator generic, ce va fi ulterior înlocuit cu valoarea adresei de IP şi a portului. Astfel, blocul care preia datele de la interfaţa web va avea ca identificator şirul de caractere __CALCULOS_TO_FBDK__, blocul care transmite rezultatul către instalaţie va avea identificatorul __FBDK_TO_PLANT__, iar în eventualitatea că există un bloc ce preia din instalaţie valorile parametrilor de proces, acesta va avea identificatorul __PLANT_TO_FBDK__.

Structura algoritmilor executabili. Algoritmii vor putea fi disponibili sub forma unor fişiere executabile, generate în orice mediu de dezvoltare. Această variantă corespunde constituirii unor regulatoare virtuale generice, ce pot fi executate în cloud. Ca şi în cazul algoritmilor editabili, fie-

care algoritm va fi însoţit de o scurtă descriere care să detalieze funcţionalitatea îndeplinită de acesta, dacă este aplicabil numai unor anumite procese, domeniul de utilizare, denumirea şi rolul parametrilor de intrare şi de ieşire (date şi evenimente) şi eventual rezultatul aşteptat în urma implementării. Algoritmii care implementează funcţii mai complexe (de modelare, optimizare, analiză etc) vor putea fi executaţi în cloud, pentru ca apoi rezultatul să fie trimis către un echipament aflat la distanţă. Pentru ca acest lucru să fie posibil, funcţia bloc compusă specifică algoritmului va fi introdusă într-o structură de tip parametrizare comunicaţie, structură disponibilă în mai multe variante:

(a) Pentru aplicaţii care necesită execuţia unor algoritmi pe baza unor date de timp real şi transmiterea rezultatelor către instalaţie, se va ataşa o interfaţă de intrare OPC şi o interfaţă de ieşire OPC. Pentru aceasta, la configurarea algoritmului pentru execuţie în cloud, utilizatorul va trebui să specifice următoarele:

- Adresa de IP a sursei datelor;
- Numele serverului de OPC din care se face accesul datelor;
- Calea şi numele parametrilor corespunzători datelor de intrare necesare pentru execuţia algoritmului;
- Adresa de IP a destinaţiei datelor;
- Numele serverului de OPC către care se va face scrierea datelor;
- Calea şi numele parametrilor corespunzători datelor de ieşire care vor prelua valorile rezultate în urma execuţiei algoritmului.

(b) Pentru aplicaţii care necesită achiziţia unor date din proces şi stocarea acestora într-o bază de date din cloud, vor fi necesare următoarele informaţii:

- Adresa de IP a sursei datelor;
- Numele serverului de OPC din care se face accesul datelor;
- Calea şi numele parametrilor corespunzători datelor care vor fi arhivate;
- Detalii necesare identificării utilizatorului şi instalaţiei. Pe baza acestora se va crea un serviciu care va gestiona achiziţia şi stocarea datelor într-o bază de date SQL.

(c) Pentru aplicaţii de analiză asupra unor date de proces de tip istoric, disponibile în cloud, vor fi necesare următoarele informaţii:

- Detalii necesare identificării utilizatorului şi instalaţiei. Pe baza acestora se va crea un serviciu care va gestiona accesul la datele de istoric dintr-o bază de date SQL, precum şi scrierea rezultatelor în aceeaşi bază de date.

Structura fişierelor executabile, cu interfeţele de comunicaţie, va fi construită dinamic, utilizatorul adăugând în bibliotecă doar fişierul executabil.

4.2 Verificarea şi validarea algoritmilor

Verificarea şi validarea aplicaţiilor de control se referă la asigurarea funcţionării aplicaţiei conform cerinţelor de proiectare. Acestea sunt operaţii prin care se analizează conformitatea unui proces sau a unor funcţii cu nişte cerinţe date. Principalele tehnici prin care se poate face verificarea sunt simularea şi testarea modelului [38]. Simularea analizează răspunsul sistemului la un set unic de parametri de intrare, iar testarea modelului este o metodă prin care se atestă corectitudinea modelului pentru diverse situaţii.

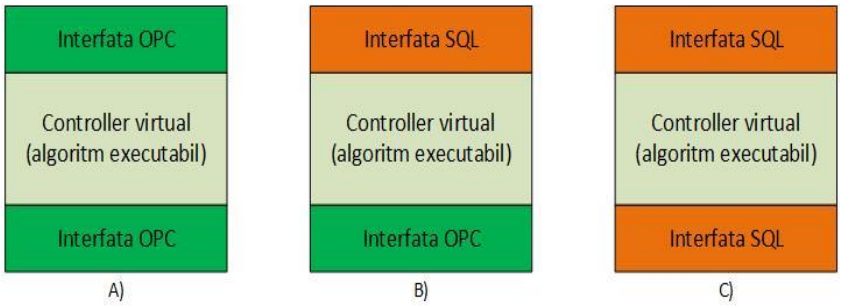

Figura 11: Structura algoritmilor executabili.

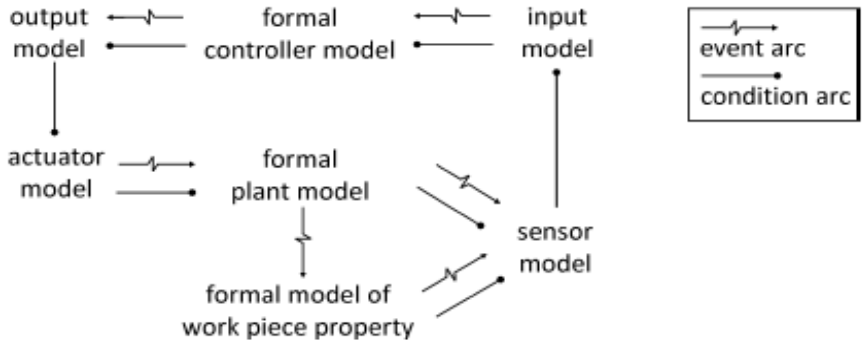

Figura 12: Schema modelării în buclă închisă a unei aplicaţii de control [16].

Cea mai uşoară metodă de verificare a unui algoritm prin metoda simulării constă în stabilirea unui set fixat de parametri de intrare şi analiza răspunsului obţinut. Analiza răspunsului se poate face fie raportat la descrierea analitică a algoritmului, fie prin implementarea aceluiaşi algoritm într-un alt mediu de dezvoltare, precum MATLAB, şi compararea răspunsurilor. Întrucât modurile de implementare în IEC 61499 şi MATLAB diferă, comparaţia se poate face de obicei asupra rezultatului final. Dacă acesta nu corespunde, se poate apela la verificarea unor subcomponente pornind de la formulele analitice ale algoritmului. În [16] se abordează problema verificării şi validării aplicaţiilor de control pornind de la testarea modelului procesului. Şi în acest caz există două situaţii: se poate construi modelul regulatorului astfel încât să se obţină răspunsul dorit în buclă închisă, sau se poate folosi un model de regulator şi în urma analizei răspunsului procesului în buclă închisă să se verifice respectarea cerinţelor sistemului. Cea de-a doua variantă poate presupune mai multe iteraţii, cu ajustarea corespunzatoare la fiecare pas a modelului regulatorului.

Din punctul de vedere al utilizării funcţiilor bloc bazate pe standardele IEC 61131-3 şi IEC 61499, modelarea procesului şi a regulatorului pot necesita un efort destul de ridicat. De asemenea, este necesară identificarea unor reguli de transformare între aplicaţia bazată pe funcţii bloc şi nişte modele formale, atât pentru partea de control a execuţiei, cât şi pentru controlul algoritmilor. Schema obţinerii unui model în buclă închisă este prezentată în Fig. 12.

Pentru modelarea cerinţelor sistemului se pot folosi atât metode cunoscute precum reţele Petri, Timed Automata sau UML (Unified Modelling Language), cât şi metode mai noi precum Timed

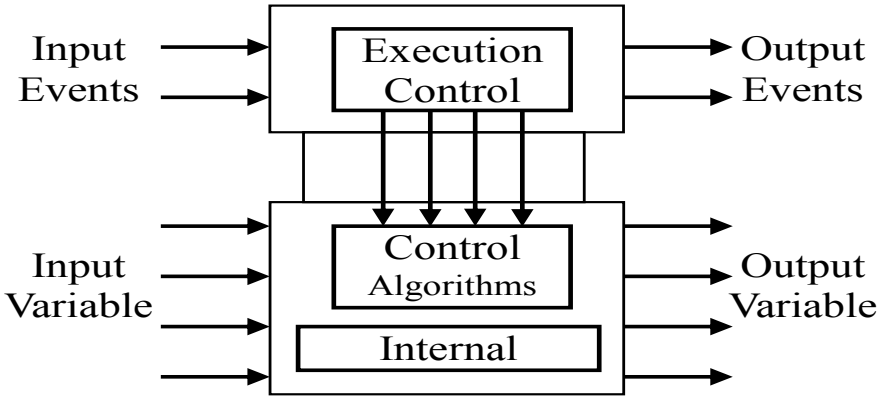

Figura 13: Reprezentarea unei funcţii bloc conform standardului IEC 61499.

Computation Tree Logic (TCTL) sau Computation Tree Logic (CTL). Pentru acestea există şi editoare grafice, în care specificaţiile sunt construite sub formă de diagrame şi formule adaptate metodei de modelare aleasă. Astfel modelul răspunsului în buclă închisă al sistemului poate fi analizat relativ la modelul formal al cerinţelor. Astfel de metode de verificare şi validare a aplicaţiilor de control au fost implementate pe diverse platforme de testare, de cele mai multe ori academice. Pentru a minimiza efortul verificării şi validării aplicaţiilor ce utilizează funcţii bloc bazate pe standardele IEC 61131-3 şi IEC 61499, se poate folosi aplicaţia MATLAB Simulink pentru modelarea procesului şi a regulatorului, urmând ca apoi să se aplice o metodă de transformare în standardul dorit. O metodă de transformare din Simulink în IEC 61499 este prezentată în [39]. Această metodă poate fi cu uşurinţă adaptată pentru a fi aplicată în cadrul standardului IEC 61131-3. Totuşi, aceasta ar presupune un efort suplimentar, care nu este necesar în condiţiile în care pot fi folosite extensii ale Simulink, precum PLC Coder sau PLC link, pentru generarea automată a codului conform acestui standard. Metoda de transformare din [39] funcţionează pe principiul mapării directe a funcţiilor bloc din Simulink pe cele specifice standardului IEC 61499. Astfel, pentru fiecare funcţie bloc din Simulink este creată o funcţie în IEC 61499. De asemenea, pentru variabilele de intrare există o corespondenţă de unu la unu. Din punctul de vedere al variabilelor interne, acestea sunt mapate în IEC 61499 ca variabile de intrare, pentru a permite modificarea uşoară a valorilor acestora în mediile de dezvoltare specifice. Funcţia de control a execuţiei este mapată la un bloc de tip Stateflow din Simulink, prin crearea corespondenţei directe între stările, condiţiile, tranziţiile şi algoritmii de control din cele două reprezentări.

4.3 Utilizarea reţelelor de funcţii bloc în conducerea proceselor

Elementul de proiectare de bază al arhitecturii IEC 61499 este blocul funcţional, sau *funcţia bloc* (FB). Funcţiile bloc pot fi utilizate pentru descrierea logicii de control descentralizate şi a proprietăţilor dispozitivelor, cât şi a interfeţelor acestora, după se ilustrează în Fig. 13.

Pentru a determina precis comportarea unui dispozitiv interconectat într-o aplicaţie distribuită, este important să fie cunoscute regulile execuţiei unei funcţii bloc, adică, semantica [7]. Standardul IEC 61499 defineşte semantica pentru funcţii bloc de bază şi compozite, cât şi pentru reţelele acestora. Funcţiile bloc au definite interfeţe pentru intrări şi ieşiri de evenimente şi date. Intrările de tip eveniment sunt utilizate pentru a activa o funcţie bloc. Comportarea unei funcţii bloc de bază este determinată de o maşină de stare, numită Diagrama de Control al Execuţiei (Execution Control

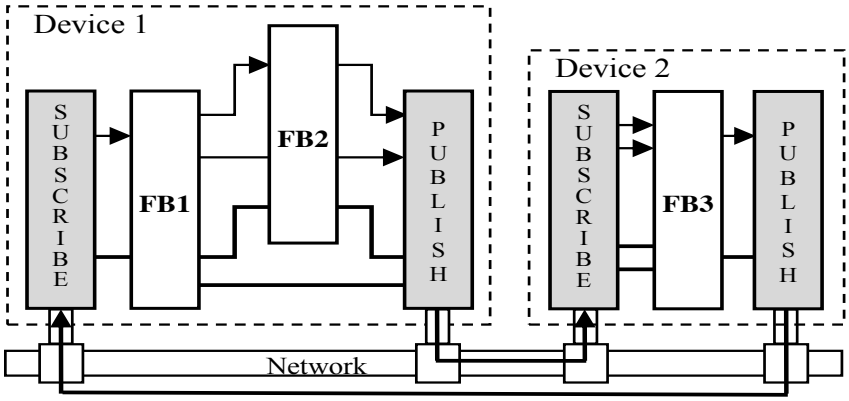

Figura 14: Conexiunea unidirecţională a două dispozitive IoT utilizând funcţii bloc.

Chart — ECC). Stările unei ECC pot avea asociate acţiuni, constând fiecare din invocarea unui algoritm şi emiterea unui eveniment de ieşire. Instanţele unei funcţii bloc pot fi conectate cu alte funcţii bloc, formând reţele de funcţii bloc. Semantica execuţiei reţelei este dată de definirea fluxului de date între instanţele funcţiilor bloc. Reţelele de funcţii bloc sunt privite ca un model general al sistemelor de conducere automată, atât centralizate, cât şi distribuite. În sistemele distribuite, instanţele funcţiilor bloc incluse într-o reţea pot fi considerate ca procese independente. Comunicaţia dintre ele este modelată prin transmiterea de evenimente şi date. Standardul IEC 61499 furnizează două modele generice de comunicaţie: PUBLISH/SUBSCRIBE, pentru comunicaţia unidirecţională, şi CLIENT/SERVER pentru comunicaţia bidirecţională. Dispozitivelor distribuite le sunt asociate funcţii bloc de aplicaţie, iar conexiunilor de evenimente sau date între dispozitive li se asociază funcţii bloc de comunicaţie. Pentru citirea etichetelor unui dispozitiv RFID este necesară doar comunicaţie unidirectională. Figura 14 arată schema bloc a distribuirii unei aplicaţii între două dispozitive “Internet of Things” (IoT). Conexiunile care traversează limitele dispozitivelui sunt reprezentate prin funcţii bloc de comunicaţie.

De remarcat că arhitectura de funcţii bloc a standardului IEC 61499 poate furniza soluţii pentru reprezentarea relaţiei logice între servicii la nivelul sistemului şi, în particular, oferi o reprezentare adecvată pentru reconfigurarea serviciilor pe durata de viaţă a sistemului.

De fapt, se poate demonstra complementaritatea dintre SOA (Software Oriented Architecture) şi IEC 61499:

1. în accepţiunea SOA, functionalităţile sunt încapsulate în servicii care comunică între ele doar prin mecanismul de transmisie de mesaje;

2. reprezentarea prin reţele de funcţii bloc corespunde perfect rolului de descriere a serviciilor (şi, de asemenea, a relaţiilor dintre ele). Astfel, tipurile de funcţii bloc pot fi considerate ca definiţii de tipuri de servicii. Transmiterea de mesaje între servicii este reprezentată de conexiunile dintre funcţiile bloc.

Există două tipuri de conexiuni în standardul IEC 61499: de evenimente şi de date. Intrările şi ieşirile de date trebuie asociate cu evenimente de intrare şi ieşire, cu scopul de a transmite valori în şi din funcţiile bloc. În accepţiunea SOA, fiecare eveniment de conexiune este asociat unui tip de mesaj. Variabilele de date asociate acestui eveniment sunt utilizate ca parametri de intrare ai mesajului. În afară de IoT şi integrarea RFID, prin intermediul funcţiilor bloc se poate realiza, de

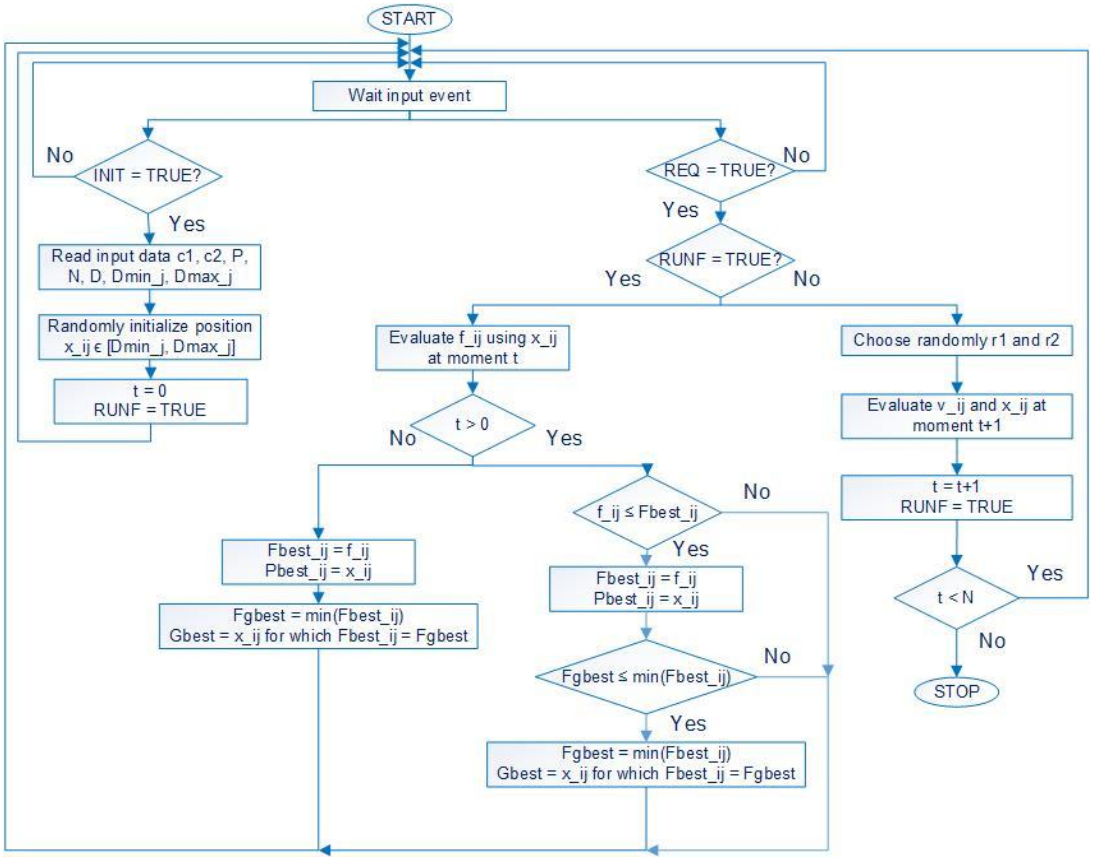

Figura 15: Diagrama algoritmului PSO.

asemenea, accesul la cloud. Conform IEC 61499, funcţiile bloc sunt proiectate ca să fie componente reutilizabile, distribute de la nivelul de control inferior până la nivelul de control cel mai înalt.

Adaptabilitatea la noi cerinţe este realizată prin includerea cunoaşterii la nivelul superior. Este normal ca operaţiile realizate prin funcţii bloc să poată fi considerate ca servicii cloud [7]. De accea, la nivel logic, funcţionalitatea este încapsulată în elemente de bază numite servicii, care pot invoca alte servicii cu scopul de a realiza o sarcină.

5 Implementarea algoritmilor standardizaţi

5.1 Algoritmul PSO

5.1.1 Proiectarea algoritmului PSO utilizând funcţii bloc IEC 61499

Considerând o problemă de minimizare a unei funcţii f, în Fig. 15 sunt ilustraţi paşii necesari pentru implementarea algoritmului, luând în considerare mecanismul de execuţie a funcţiilor bloc IEC 61499.

În Fig. 15, INIT şi REQ sunt evenimente de intrare care lansează execuţia secvenţelor corespunzătoare. Notaţia *_ij reprezintă valoarea variabilei *, pentru $i = 1, \ldots, P$, $j = 0, \ldots, D$, unde P este numărul total de particule, iar D numărul de variabile, sau dimensiunea problemei. Dmin_j şi Dmax_j sunt capetele intervalului în care ar trebui efectuată căutarea pentru fiecare dimensiune.

Contorul t identifică iteraţia curentă. RUNF este o variabilă necesară pentru separarea execuţiei algoritmului de partea care face evaluarea funcţiei la poziţia curentă. Aceasta permite scrierea unui algoritm generic, reutilizabil, ce nu depinde de funcţia obiectiv pentru care este utilizat. Fbest reprezintă un vector ce reţine valorile minime ale funcţiei obiectiv la momentul t iar Pbest conţine poziţiile corespunzătoare pentru care s-au obţinut acele valori. Fgbest stochează optimul global iar Gbest reprezintă valorile j corespunzătoare poziţiilor pentru care s-a obţinut acel optim. v_ij şi x_ij sunt evaluate conform ecuaţiilor din secţiunea 2.2. De asemenea, toate celelalte variabile au specificaţia conform definiţiei de acolo. Algoritmul se opreşte când se atinge numărul maxim de variabile. Ieşirile algoritmului sunt date de variabilele Gbest şi Fgbest.

5.1.2 Implementarea algoritmului

Dezvoltarea şi implementarea algoritmului s-a făcut utilizând mediul de dezvoltare FBDK, versiunea 2.1. Algoritmul a fost dezvoltat sub forma unei funcţii bloc având structura prezentată în Fig. 16. Pentru simplificare, s-a luat în considerare o singură dimensiune ($D = 1$). Datorită faptului că FBDK nu permite definirea vectorilor de lungime variabilă, s-a considerat lungimea maximă recomandată de 60. Acest lucru va avea un impact minim numai asupra spaţiului de stocare. Performanţa de execuţie nu va fi afectată întrucât parcurgerea vectorilor se face în funcţie de numărul de particule P definit ca intrare. X_F este valoarea lui X trimisă spre evaluare către funcţia obiectiv. F este valoarea funcţiei obiectiv pentru acel X_F. K este o variabilă internă ce ţine evidenţa elementului curent din vectorul de poziţie X ce trebuie trimis pentru evaluare către f. Toate celelalte variabile au semnificaţiile detaliate anterior.

Diagrama ECC a funcţiei bloc PSO este ilustrată în Fig. 17. Evenimentul INIT este folosit pentru a lansa funcţia INIT care iniţializează variabilele algoritmului, pentru ca apoi să trimită un eveniment de ieşire INITO. Aceasta secvenţă corespunde primei ramuri din diagrama din Fig. 15. La recepţionarea unui eveniment REQ, algoritmul verifică valoarea variabilei RUNF. Dacă aceasta este falsă, atunci se execută funcţia STEP ce implementează un pas al algoritmului prin calcularea vectorilor de viteză şi de poziţie. Dacă se atinge numărul maxim de iteraţii, variabila DONE ia valoarea TRUE, ceea ce activează evenimentul de ieşire END_F. Aceasta corespunde ultimei ramuri a diagramei din Fig. 15. Dacă RUNF are valoarea TRUE înseamnă că trebuie să se execute funcţia CALC_F. Aceasta corespunde secţiunii rămase din diagrama din Fig. 15. După cum s-a menţionat anterior algoritmul se doreşte a fi reutilizabil şi independent de funcţia minimizată. Din acest motiv, funcţia CALC_F utilizează o variabilă k internă, care variază de la 0 la P şi o varibilă de ieşire, X_F, ce trimite elementul k al vectorului de poziţie X către funcţia f. Evaluarea lui f(X_F) reprezintă variabila F de la intrarea funcţiei bloc şi este comparată cu Fbest[k]. Dacă noua valoare este mai mică, atunci X_F şi F sunt stocate în Pbest[k] şi respectiv Fbest[k]. Când variabila k ajunge la sfârşitul vectorului de poziţie (k=P), atunci RUNF ia valoarea FALSE, permiţând astfel funcţiei bloc să execute un nou pas al algoritmului. După parcurgerea funcţiei STEP, RUNF ia valoarea TRUE, variabila t este incrementată, iar variabila k resetată.

Atunci când se atinge numărul maxim de iteraţii se generează evenimentul de ieşire END_F iar algoritmul intră într-o stare de aşteptare până la apariţia unui nou eveniment INIT. Cei trei algoritmi componenţi sunt detaliaţi mai jos. Întrucât era necesară utilizarea unor formule matematice mai complexe, s-a optat pentru reprezentarea utilizând Java.

La iniţializarea algoritmului, vectorul poziţiilor ia valori aleatoare în domeniul de căutare [Dmin, Dmax]. Vectorul iniţial al vitezelor este nul. Variabila RUNF are la început valoarea TRUE, următorul pas fiind astfel execuţia algoritmului CALC_F pentru calculul valorii lui F pentru vectorul de poziţie iniţial. Variabila k reţine poziţia din vectorul X pentru care trebuie calculată valoarea funcţiei. Variabila t este utilizată pentru a contoriza numărul de iteraţii efectuate.

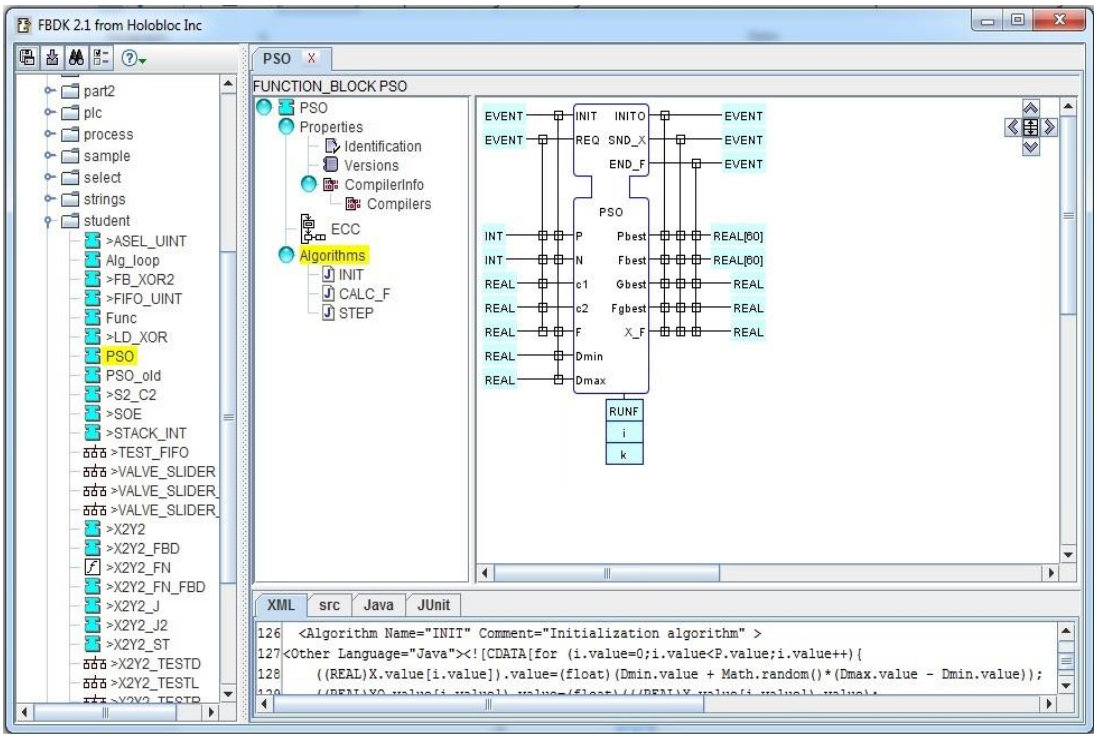

Figura 16: Structura algoritmului.

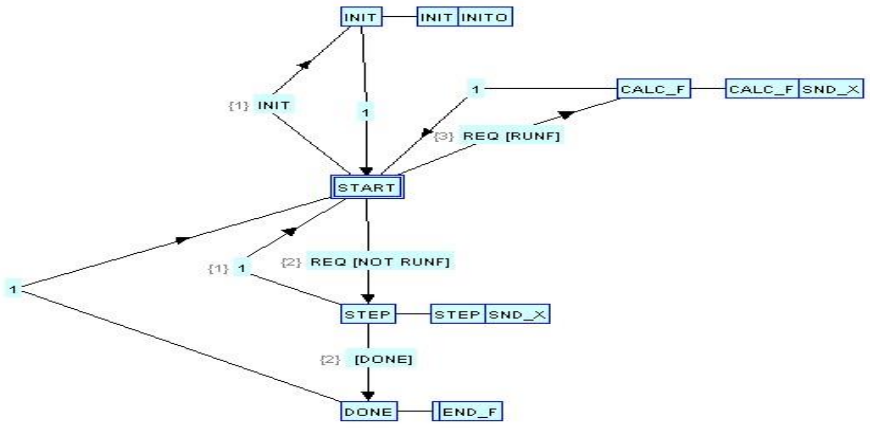

Figura 17: Diagrama de execuţie ECC pentru algoritmul PSO.

```
INIT
for (i.value=0;i.value<P.value;i.value++){
        ((REAL)X.value[i.value]).value=(float)(Dmin.value +
                Math.random()*(Dmax.value - Dmin.value));
        ((REAL)V.value[i.value]).value= 0;
}

k.value=0;
```

```
RUNF.value = true;
F.value = 0;
X_F.value = 0;
t.value=0;
DONE.value=false;
```

Algoritmul CALC_F are ca scop calcularea lui F pentru fiecare element al vectorului de poziţie f la fiecare iteraţie şi căutarea punctului optim. Primele valori ale lui F se obţin după execuţia algoritmului INIT. Vectorul de poziţie este parcurs prin incrementarea variabilei k şi furnizarea valorii curente a acestuia prin intermediul variabilei X_F, pentru a putea fi calculată valoarea funcţiei. La prima iteraţie (t = 0) algoritmul reţine valorile lui F, precum şi cele ale lui X_F. Dacă s-a ajuns la sfârşitul vectorului de poziţie (k = P) pentru iteraţia curentă t, se calculează minimul vectorului Fbest şi se reţine valoarea în variabila Fgbest. De asemenea, valoarea lui X pentru care a fost obţinut acel minim se reţine în variabila Gbest. Apoi variabila RUNF ia valoarea FALSE pentru a permite calcularea unui nou vector de poziţie, iar pointerul se poziţionează la începutul acestui vector (k = 0).

```
CALC_F
if( (k.value>1) ){
    k_ant.value =(short)( k.value-1);
    if( ((t.value==0) ) ){
        ((REAL)Fbest.value[k_ant.value]).value=(float)(F.value);
        ((REAL)Pbest.value[k_ant.value]).value=(float)(X_F.value);
    }
}
if( (k.value==P.value) ){
    if( (t.value==0) ){
        Fgbest.value=(float)(((REAL)Fbest.value[0]).value);
        Gbest.value=(float)(((REAL)Pbest.value[0]).value);
    }
    for (i.value=0;i.value<P.value;i.value++){
        if( (((REAL)Fbest.value[i.value]).value<=Fgbest.value) ){
            Fgbest.value=(float)(((REAL)Fbest.value[i.value]).value);
            Gbest.value=(float)(((REAL)Pbest.value[i.value]).value);
        }
    }
    RUNF.value = false;
    k.value=(short)(0);
}else{
    X_F.value=(float)(((REAL)X.value[k.value]).value);
    k.value=(short)(k.value+1);
}
```

Algoritmul STEP realizează calculul vectorilor de poziţie X şi de viteză V de la iteraţia curentă. Pentru aceasta sunt generaţi întâi parametrii aleatori r1 şi r2. Variabila ao este folosită ca precalcul prentru simplificarea formulei pentru vectorul V. Vectorii V şi X sunt calculaţi conform formulelor (6) pe baza valorilor anterioare, a parametrilor r1, r2, a lui ao şi a vectorului Pbest. După finalizarea calculului, variabila RUNF ia valoarea TRUE pentru a se calcula valoarea lui f pentru noul vector de poziţie. Variabila t este incrementată pentru a putea ţine evidenţa numărului de iteraţii efectuate. Atunci când este atins numărul maxim de iteraţii (t = N), algoritmul se opreşte şi se generează un eveniment de ieşire END_F prin intermediul variabilei booleene DONE.

```
STEP
if( (t.value<N.value) ){
r1.value=(float)(Math.random()*c1.value);
r2.value=(float)(Math.random()*c2.value);

for (i.value=0;i.value<P.value;i.value++){
    ao.value=(float)(Gbest.value - ((REAL)X.value[i.value]).value);
    ((REAL)V.value[i.value]).value=(float)(
              ((REAL)V.value[i.value]).value +
            + r1.value*(((REAL)Pbest.value[i.value]).value -
            - ((REAL)X.value[i.value]).value)+r2.value*ao.value);

    ((REAL)X.value[i.value]).value=(float)(
              ((REAL)X.value[i.value]).value +
            + ((REAL)V.value[i.value]).value);
    }
    RUNF.value=true;
    t.value=(short)(t.value+1);
}else{
    DONE.value=true;
    t.value=(short)(0);
}
```

5.1.3 Testarea şi validarea algoritmului

Testarea algoritmului s-a efectuat folosind o configuraţie ce conectează funcţia bloc PSO de funcţia obiectiv denumită Test_func, după cum se poate vedea în Fig. 18. Au fost adăugate câteva variabile de ieşire pentru a evalua mai bine execuţia algoritmului.

Pentru execuţie, s-au folosit următorii parametri de intrare: dimensiunea populaţiei $P = 20$, numărul de iteraţii $N = 30$, $c_1 = c_2 = 2$, Dmin = 0, Dmax = 15. Funcţia obiectiv de test, Test_func, implementează următoarea ecuaţie,

$$\text{out} = f(x) = x^2 + x + 1. \tag{22}$$

S-a analizat ieşirea algoritmului PSO, implementat în IEC 61499, pornind cu o populaţie iniţială aleatoare la momentul $t = 0$. Fig. 19 ilustrează evoluţia vectorului Pbest, care reţine valorile cele mai bune pentru fiecare particulă, precum şi poziţionarea şi valoarea celei mai bune valori globale în acest vector, la diferite momente de timp.

Mişcarea particulelor acoperă o suprafaţă mai mare la începutul algoritmului şi tinde să stagneze pe măsură ce valoarea Gbest se apropie de valoarea optimă. În funcţie de precizia dorită, valoarea minimă a funcţiei f, Fgbest = 0.7500, a fost obţinută după primele 4 iteraţii utilizând Gbest = -0.4971. Pe măsură ce algoritmul a continuat, cea mai bună precizie a fost obţinută în 28 iteraţii şi este reprezentată de Gbest = -0.5004, Fgbest = 0.7500. Acelaşi algoritm a fost implementat şi executat în MATLAB, utilizat la scară largă pentru programarea unor aplicaţii avansate şi pentru calcule numerice complexe, pentru a putea compara performanţele şi rezultatele obţinute din cele două medii de programare diferite. Soluţia algoritmului implementat în MATLAB, obţinută după 14 iteraţii, a fost Gbest = -0.4998, Fgbest = 0.7500. Diferenţa între vitezele de atingere a optimului se datorează populaţiei iniţiale diferite, precum şi valorilor parametrilor r_1 şi r_2 aleşi aleator la fiecare iteraţie.

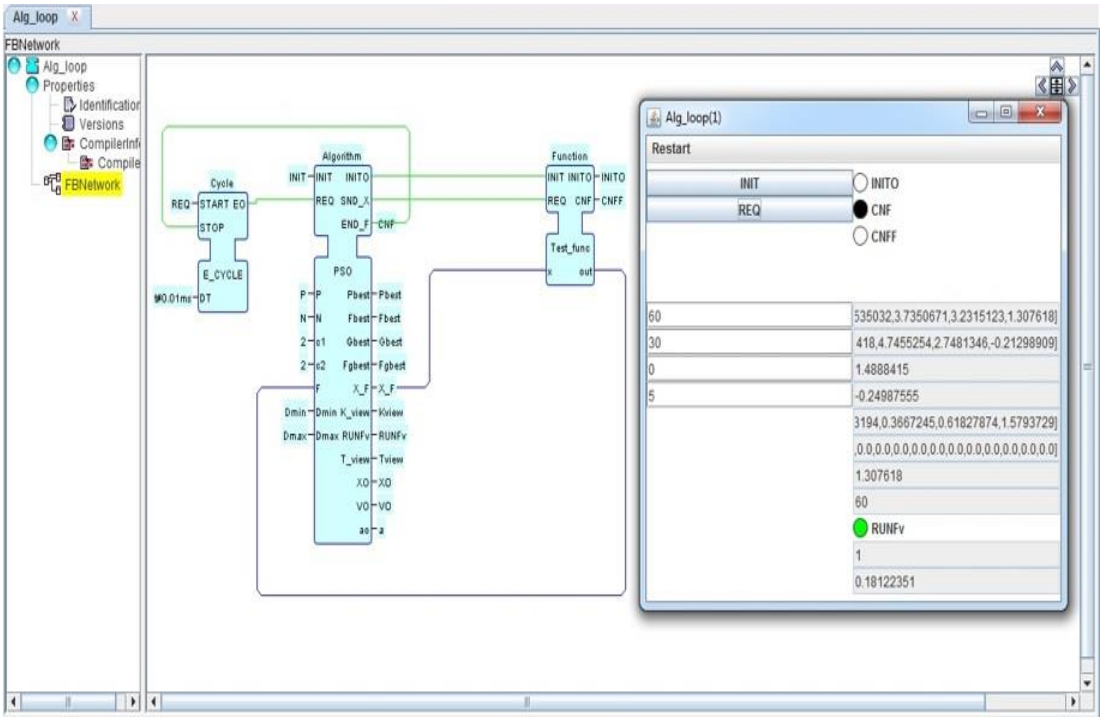

Figura 18: Configuraţia de testare a algoritmului.

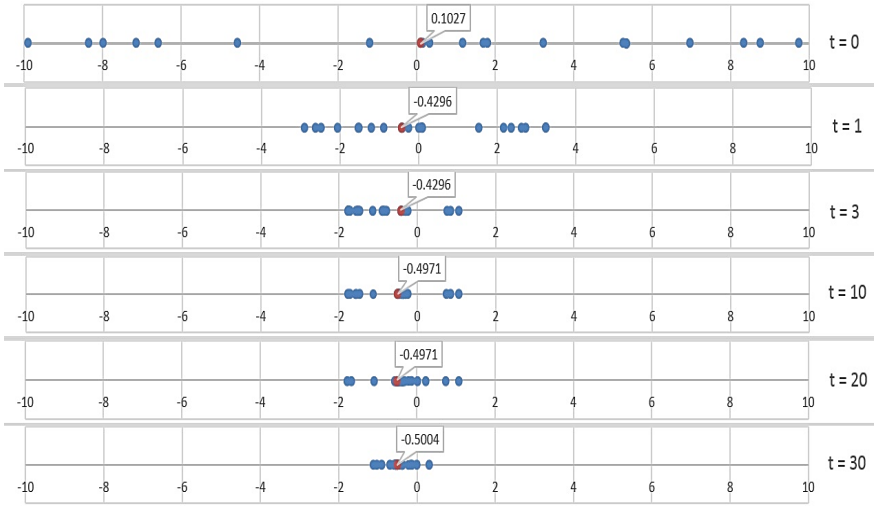

Figura 19: Mişcarea particulelor către valoarea optimă globală.

5.1.4 Exemplu de utilizare offline

Algoritmul poate fi utilizat offline pentru optimizarea unor funcţii cu mai mult de o variabilă, cu condiţia ca variabilele să fie independente între ele. Un exemplu de astfel de aplicaţie pentru o

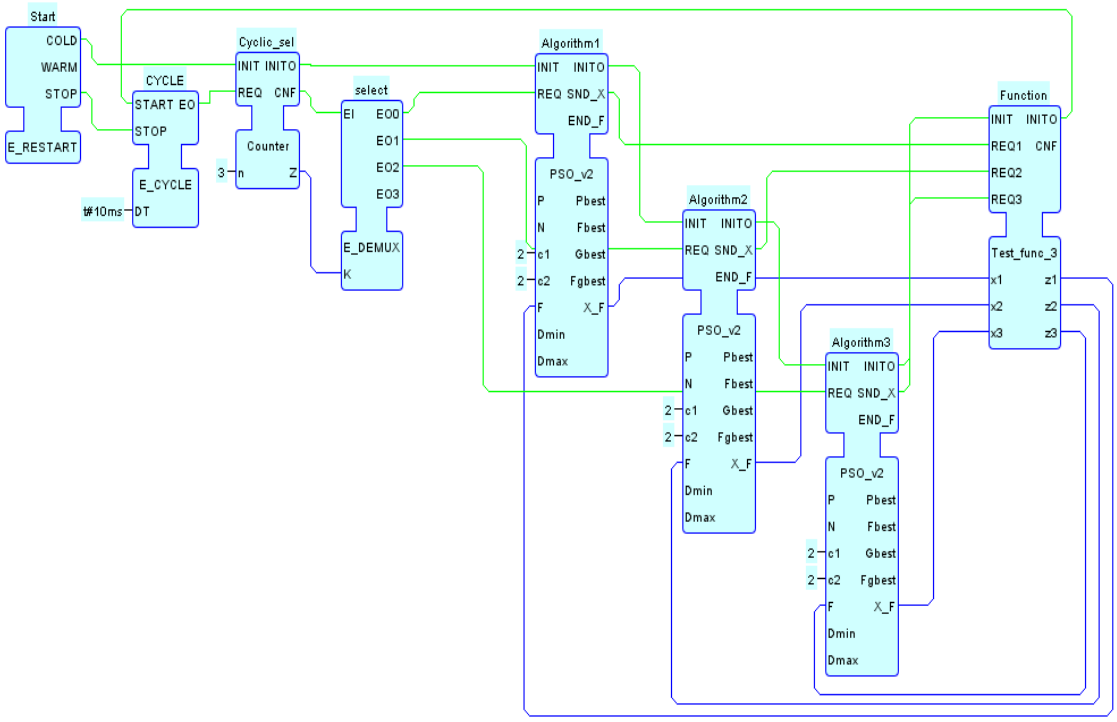

Figura 20: Optimizarea unei funcţii cu trei variabile.

funcţie cu trei variabile este ilustrat în Fig. 20. Datorită genericităţii algoritmului PSO, diferite instanţe ale lui pot fi folosite pentru fiecare ramură a funcţiei. S-a ales o structură de execuţie ciclică în care se generează un eveniment REQ pentru execuţia unui pas al unei ramuri la fiecare ciclu program. În acest caz se vor alege acelaşi număr de particule şi acelaşi număr de iteraţii în execuţia celor trei instanţe ale algoritmului. Blocul Cyclic_sel este utilizat ca numărător de la 0 la numărul total necesar de evenimente şi comandă blocul Select, responsabil pentru activarea execuţiei fiecărui algoritm. Funcţia f se execută conform diagramei ECC, pe baza evenimentului de intrare care este activat. Astfel, prin activarea evenimentului REQ1 blocul va executa funcţia f(x1) şi va trimite rezultatul către z1.

5.2 Structurarea unui algoritm în format executabil

Pentru asigurarea posibilităţii de execuţie a unui set cât mai general de algoritmi a fost creat un cadru de adaptare dinamică a execuţiei la orice format disponibil. Astfel, orice algoritm prevăzut sub formă de fişier executabil va putea fi rulat pe platformă prin adăugarea dinamică la acesta a unor interfeţe de interconectare cu procesul. Aplicaţia implementată are rolul de a se conecta la un server OPC, a prelua date de pe server, a rula un executabil cu datele de pe serverul OPC ca date de intrare, a prelua apoi datele de ieşire ale executabilului, şi, în final, a scrie pe serverul OPC datele de ieşire. În acest scop, s-au folosit biblioteci suplimentare din cadrul proiectului Utgard de la openSCADA. Interfaţa suportată în momentul de faţă este OPC DA 2.0. Pentru a se conecta la server se creează întâi un obiect de tip ConnectionInformation în care se depun datele de identificare: setHost(), setUser(), setPassword() şi setClsid(). Se creează un şir de caractere pentru a stoca id-ul locaţiei

datelor pe server şi apoi un obiect de tip Server cu datele de identificare de mai sus. Cel de-al doilea argument, Executors.newSingleThreadScheduledExecutor(), creează un fir de execuţie independent de program, care este folosit de fiecare dată când se vrea accesarea datelor de pe server. Codul sursa este detaliat mai jos.

```
final ConnectionInformation ci = new ConnectionInformation();
ci.setHost("localhost");
ci.setUser("SISLAPTOP");
ci.setPassword("sis");
ci.setClsid("6E6170F0-FF2D-11D2-8087-00105AA8F840");
final String itemId = "Channel_1.Device2.Input";
final Server server = new Server(ci, Executors.newSingleThreadScheduledExecutor());
```

Se conectează apoi la server şi se adaugă un obiect de tip SyncAccess(). Cu acest obiect se actualizează datele citite de pe server la perioada menţionată în cel de-al doilea argument, primul argument având rolul de a specifica serverul. Se accesează apoi serverul cu rutina addItem(), prin care se specifică id-ul locaţiei datelor de pe server, secvenţa de cod ce se va executa la fiecare ciclu, reprezentată de un obiect de tipul DataCallback(), în care se specifică funcţia changed(). Variabila în care se stochează valoarea de pe server este de tip String, iar funcţia care returnează valoarea de pe server este getValue().getObjectAsUnsigned().getValue(). Pentru a asigura o singură execuţie a funcţiei se apelează procedura unbind(), ce opreşte firul de execuţie care citeşte de pe server.

```
server.connect();
final AccessBase access = new SyncAccess(server, 100);
access.addItem(itemId, new DataCallback() {
   public void changed(Item item, ItemState state) {
      try{
         if (state.getValue().getType() == JIVariant.VT_UI4) {
            varb.nr = ""+
               state.getValue().getObjectAsUnsigned().getValue();
            System.out.println(">>read<<");
            System.out.println(varb.nr);
            varb.ct2=1;
         } else {
            System.out.println("<<< " + state + " / value = " +
               state.getValue().getObject());
         }
      } catch (JIException e) {
         e.printStackTrace();
      }
      try {
         access.unbind();
      } catch (JIException e) {
         // TODO Auto-generated catch block
         e.printStackTrace();
      }
   }
});
```

Pentru a executa programul cu variabila proaspăt citită, se mai creează un fir de execuţie ce aşteaptă după firul de mai sus. Acest lucru este necesar deoarece, executându-se în paralel, executabilul este rulat înainte de a citi valoarea de pe server. Acest lucru se poate face şi cu un ciclu while infinit, dar cu o condiţie de încetare atunci când s-a citit valoarea. Pentru a verifica dacă s-a citit

variabila sau nu, s-a creat o variabilă care iniţial este 0, dar devine 1 după citire. Firul de execuţie este pus pe o condiţie de if, care este validă dacă variabila este 1. Firul de execuţie este creat tot cu ajutorul unui obiect de tip Executors.newSingleThreadScheduledExecutor(). Firul este lansat în execuţie cu ajutorul funcţiei scheduleWithFixedDelay(), care primeşte ca argumente un obiect de tip Runnable(), ce conţine funcţia run() redefinită, două argumente ce vor avea rol în executarea cu întârziere a funcţiei run() şi unitatea de timp a valorilor specificate în argumentele anterioare. Pentru a rula şi prelua datele executabilului s-au folosit patru obiecte, de tipul ByteArrayOutputStream(), PumpStreamHandler(), DefaultExecutor() şi, respectiv, CommandLine(). Cele patru obiecte ajută la rularea executabilului şi capturarea ieşirii acestuia. Prima dată se creează un obiect din primul tip. Se creează apoi cel de tipul CommandLine(), care se iniţializează cu comanda prin funcţia parse(). Se creează un obiect de al treilea tip în care se specifică prin argument tipul outputStream. Acest obiect se va folosi pentru a schimba tipul de stream. Se schimbă apoi tipul de stream cu funcţia exec.setStreamHandler(). Pentru a rula executabilul se apelează la linia de comandă din sistemul de operare (aici Windows). Se execută apoi în linia de comandă executabilul prin funcţia exec.execute(). Se pune ieşirea într-o variabilă de tip şir de caractere. Pentru a identifica numărul, se elimină toate caracterele de după număr şi se ia fiecare caracter şi se scade din caracterul 0. Astfel se obţine numărul care este pus într-o varabilă de tip int. Se opreşte apoi firul de execuţie cu funcţia shutdownnNow().

```
ScheduledExecutorService writeThread =
    Executors.newSingleThreadScheduledExecutor();
writeThread.scheduleWithFixedDelay(new Runnable() {
    @Override
    public void run() {
        if(varb.ct2==1){
            System.out.println(">>call SYS<<");
            String cmd="cmd /c call
                C:\\Users\\SISLAPTOP\\Desktop\\usr\\OPC\\new\\a.bat "+
                varb.nr;
            ByteArrayOutputStream outputStream = new
                ByteArrayOutputStream();
            CommandLine commandline = CommandLine.parse(cmd);
            DefaultExecutor exec = new DefaultExecutor();
            PumpStreamHandler streamHandler =
                new PumpStreamHandler(outputStream);
            exec.setStreamHandler(streamHandler);
            String s="";
            try {
                exec.execute(commandline);
                s=outputStream.toString();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            int i;
            int n=0;
            s=s.substring(0,s.length()-2);
            for(i=0;i<s.length();i++){
                n=n*10+s.charAt(i)-'0';
                //System.out.println(s.charAt(i)+"<"+n);
            }
            varb.n=n;
```

```
            System.out.println(n);
            varb.donep=0;
            writeThread.shutdownNow();
        }
    }
},0, 1000 , TimeUnit.MILLISECONDS);
```

Scrierea valorii de ieşire pe server se face cu un ciclu while care aşteaptă execuţia firului de mai sus. Acest fir setează o variabilă 0 când s-a executat. Variabila este iniţial 1, condiţia testului if nepermiţând să parcurgă codul. Dacă s-a rulat executabilul, se creează întâi un obiect de tip JIVariant() care are ca argument de intrare numărul în care s-a stocat ieşirea executabilului. Acest obiect este folosit pentru a scrie valoarea pe server cu funcţia write(), care are ca argument de intrare acest obiect. Ciclul while se întrerupe cu execuţia instrucţiunii break. În final se deconectează de la server. Înainte de a se scrie pe server s-a creat un grup şi s-a adăugat variabila de ieşire, după care s-a permis accesul la server prin funcţia bind(). Acest cod a a fost implementat astfel:

```
final Group group = server.addGroup("test");
final String itemId2 = "Channel_1.Device_1.Output";
final Item item = group.addItem(itemId2);
access.bind();
Cod sursa scriere pe server:
while(true){
    if(varb.donep==0){
        final JIVariant value = new JIVariant(varb.n);
        try {
            item.write(value);
            System.out.println(">>ready<<");
        } catch (JIException e) {
            e.printStackTrace();
        }
        break;
    }
}
server.disconnect();
```

6 Elaborare arhitectură cloud şi specificaţii

6.1 Selectarea tehnologiilor

Componenta cloud a platformei CALCULOS reprezintă motorul principal al execuţiei şi gestionării sarcinilor utilizatorilor. În identificarea soluţiei optime ce va fi utilizată, s-a analizat atât posibilitatea folosirii standardului IEC 61499 pentru reprezentarea funcţiilor, cât şi posibilitatea de adăugare a unor algoritmi deja dezvoltaţi, pe baza unei metodologii proprii utilizatorului, ce pot fi introduşi în bibliotecă sub formă de fişiere executabile. Se creşte astfel flexibilitatea tipurilor de algoritmi ce pot fi gestionaţi, urmărind, în acelaşi timp, o structură similară de funcţii bloc. În alegerea celei mai bune infrastructuri cloud care să ofere flexibilitatea şi funcţionalitatea necesare pentru dezvoltarea unor aplicaţii de control avansat, am luat în considerare disponibilitatea alocării dinamice a resurselor de calcul, fiabilitatea stocării datelor, scalabilitatea şi posibilitatea de acces la cerere. Acestea pot fi obţinute optim prin utilizarea unei infrastructuri PaaS. Există câteva soluţii publice disponibile, precum Amazon Web Services (AWS), Microsoft Azure sau Google Cloud Platform. Fiecare dintre acestea oferă soluţii diferite referitor la tehnologia utilizată pentru server, suport pentru dezvoltator

şi integrarea aplicaţiilor. Heroku este una dintre soluţiile PaaS cele mai populare, creată de Amazon EC2, ce oferă posibilitatea încărcării, execuţiei şi managementul aplicaţiilor dezvoltate în Ruby, Node.js, Java, Phyton, Clojure, Scala sau PHP. Heroku se execută în maşini virtuale Amazon EC2 şi oferă un mediu de dezvoltare în care utilizatorul îşi poate încărca, compila sau executa codul, toate aceste operaţii făcându-se prin intermediul unor comenzi simple. Dezvoltatorul are posibilitatea sa îşi testeze şi să încarce aplicaţiile fără a necesita informaţii asupra infrastructurii. Avantajul este că Heroku administrează resursele necesare scalării proiectului. Chiar dacă Heroku oferă multe beneficii din punctul de vedere al uşurinţei în utilizare, dependenţa de Amazon EC2 nu oferă flexibilitatea şi deschiderea necesare platformei CALCULOS. De asemenea, nu oferă toleranţă la defecte, scalabilitate automată sau caracteristici de fiabilitate a datelor necesare acestei aplicaţii.

Docker este o tehnologie deschisă ce permite virtualizarea la nivelul sistemului de operare într-o manieră similară maşinilor virtuale, dar cu o încărcare a resurselor semnificativ redusă. Docker permite ca aceeaşi aplicaţie să fie încărcată în medii diferite, decuplând astfel cerinţele de infrastructură de mediul aplicaţiei. Portabilitatea oferită face posibilă integrarea acestuia în orice soluţie publică IaaS/PaaS, precum şi în infrastructuri private. În plus, Docker permite automatizarea procesului de încărcare a aplicaţiilor în cadrul containerelor, şi oferă o interfaţă de programare (API) de nivel înalt pentru managementul containerelor. Datorită performanţei acestuia, a utilizării unui număr restrâns de resurse, a simplităţii pentru împachetarea şi încărcarea aplicaţiei în interiorul containerelor, precum şi datorită suportului industrial oferit, Docker a devenit un standard popular pentru dezvoltarea de aplicaţii în cloud, în special pentru cele în care portabilitatea este un aspect deosebit de important.

În [40] se face o scurtă trecere în revistă a standardului IEC 61499, fiind prezentate principalele concepte de bază ale acestuia. Astfel, funcţiile bloc de bază (BFB) reprezintă unităţile atomice de execuţie în cadrul standardului IEC 61499. O funcţie bloc de bază este formată din două elemente, o interfaţă a funcţiei bloc (FB) şi o componentă de control a execuţiei algoritmilor (ECC), care operează peste un set de evenimente şi variabile. Executarea unei FB implică acceptarea valorilor de intrare prin intermediul interfeţei sale, prelucrarea acestor intrări cu ajutorul algoritmilor care sunt instanţiaţi de către ECC, şi apoi generarea unor valori de ieşire. O FB este încapsulată printr-o interfaţă care expune intrările şi ieşirile funcţiei bloc respective cu ajutorul porturilor de intrare şi de ieşire. Acestea pot fi clasificate ca fiind porturi pentru evenimente sau ca porturi pentru date. Comportamentul unei funcţii bloc poate fi descris ca o maşină cu stări. Aceasta reacţionează la evenimentele de intrare şi execută anumite acţiuni cu scopul de a genera ieşiri ce pot fi de tipul eveniment sau date. Un algoritm poate fi considerat ca fiind un program care utilizează variabilele de intrare, precum şi variabilele interne ale funcţiei bloc, putând fi specificat într-un limbaj de programare care este suportat de către mediul de execuţie al funcţiilor bloc. Funcţiile bloc compozite (CFB) facilitează reprezentarea unor structuri ierarhizate. CFB-urile sunt similare funcţiilor bloc de bază în sensul că şi acestea sunt încapsulate prin intermediul unor interfeţe de funcţii bloc. Cu toate acestea, spre deosebire de FB, comportamentul unei CFB este implementat printr-o reţea de funcţii bloc. Funcţiile bloc de bază şi funcţiile bloc compozite pot fi specificate pe baza tipurilor acestora (FBType). O reţea de funcţii bloc (FBN) constă în instanţe ale unor FBType-uri diferite, unde fiecare FBType poate fi instanţiat de mai multe ori. Acest concept este foarte similar cu paradigma programării orientate pe obiecte care conţine clase (analoge FBType), precum şi instanţele acestora, respectiv obiecte, ce sunt analoage cu FB. Un alt tip de funcţii bloc este reprezentat de funcţiile bloc de interfaţă (SIFB). Acestea asigură comunicarea cu serviciile furnizate de sistemul de operare sau echipamentele hardware, precum:

- Elemente de interfaţă grafică (GUI) ce sunt necesare pentru implementarea interacţiunii om-maşină (HMI);
- Servicii de comunicare (sunt posibile două moduri de comunicare: unidirecţională, imple-

mentată cu ajutorul funcţiilor bloc de tipul publish/subscribe şi bidirecţională, ce este implementată cu funcţii bloc de tipul client/server);

- Interfeţe de proces sau pentru echipamentele hardware (senzori, elemente de execuţie).

Mediile de execuţie bazate pe standardul IEC 61499 includ o gamă largă de tipuri de funcţii bloc gata implementate, respectiv elemente GUI, drivere care conectează funcţia bloc cu mediul extern şi care sunt folosite pentru controlul unor elemente de execuţie sau senzori, etc. Spre deosebire de funcţiile bloc de bază al căror comportament este descris cu ajutorul unei maşini cu stări, în cazul funcţiilor bloc de interfaţă se poate descrie comportamentul acestora utilizând diagrame de secvenţe. Un exemplu care ilustrează comunicarea între două funcţii bloc de interfaţă de tipul publish/subscribe, care efectuează transmiterea unui element de date de la SIFB-ul "publisher" la SIFB-ul "subscriber" este dat în Fig. 21. Semnificaţia elementelor din figură este detaliată în continuare:

- INIT — eveniment care iniţializează SIFB-ul;
- INITO — eveniment care indică finalizarea etapei de iniţializare a SIFB-ului;
- REQ — eveniment prin care se solicită SIFB-ului publisher transmiterea elementului de date prin intermediul reţelei;
- CNF — eveniment prin care se confirmă efectuarea cu succes a transferului de date de către publisher;
- RSP — eveniment prin care se indică procesarea cu succes a datelor de către SIFB-ul subscriber;
- IND — eveniment care indică recepţionarea datelor de către subscriber;
- QI — variabilă de tipul boolean care indică faptul că SIFB-ul trebuie să fie iniţializat (starea true) sau, din contră, că serviciul trebuie să fie terminat (starea false);
- QO — variabilă de tipul boolean care indică dacă iniţializarea s-a produs cu succes sau nu;
- ID — variabilă de tipul şir de caractere care prin care se identifică SIFB-ul, respectiv poate fi construită din adresa ip şi portul utilizat de acesta;
- SD_1 — datele care sunt transmise;
- RD_1 — datele care sunt recepţionate.

Interacţiunea dintre funcţiile bloc publish-subscribe are trei faze: stabilirea conexiunii, transferul efectiv al datelor şi deconectarea. Pentru a descrie structura şi componentele sistemului distribuit de control, elementele fizice precum microcontrolerele, PLC-urile, senzorii şi elementele de execuţie, sunt mapate sub forma unor modele logice, respectiv resurse, dispozitive şi sisteme. Dispozitivele reprezintă o abstractizare a entităţilor fizice ce au capacitatea de a executa anumite funcţii în cadrul sistemului de control şi care sunt delimitate prin intermediul unor interfeţe. Acestea pot fi considerate ca fiind componentele fizice ale unui sistem distribuit de control. Fiecare dispozitiv poate să conţină unităţi individuale care îndeplinesc o anumită funcţionalitate, corespunzător subcomponentelor sale. Pentru a satisface această cerinţă a fost creat un model logic denumit resursă. Astfel, un dispozitiv poate să conţină una sau mai multe resurse care abstractizează sarcinile executate de dispozitiv, sau funcţionalitatea sa. Resursa reprezintă unitatea funcţională independentă a unui dispozitiv ce oferă servicii aplicaţiilor care se execută în cadrul unui sistem. Aceasta include

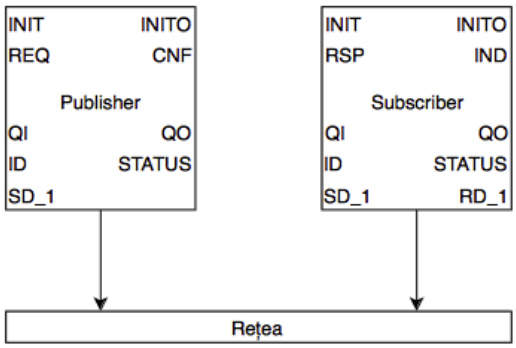

Figura 21: Funcţii bloc de interfaţă de tipul publish-subscribe.

planificarea şi executarea algoritmilor. Sarcinile executate sunt disjuncte, astfel încât o componentă fizică a sistemului (de pildă, senzor sau element de execuţie) poate fi accesată sau operată doar prin intermediul unei singure resurse.

Deoarece nu există conceptul de variabilă comună, resursele şi dispozitivele comunică prin intermediul funcţiilor bloc de interfaţă. Modelul logic al sistemului distribuit de control este reprezentat de colecţia de dispozitive interconectate şi care comunică între ele prin intermediul reţelei. Fiecare dispozitiv este capabil să execute un set de funcţii în mod independent. Coordonarea dispozitivelor se face prin mesaje transmise în cadrul reţelei, ceea ce constituie o caracteristică a unui sistem distribuit. Sistemul este format din componentele logice, un model referitor la aplicaţia care se execută în cadrul acestuia, respectiv un model al dispozitivelor şi resurselor componente. Primul model descrie logica de control, iar al doilea cum aceasta este implementată.

Modelul referitor la aplicaţia distribuită de control este compus dintr-o reţea de funcţii bloc de diferite tipuri. Pentru a putea interpreta un model creat cu ajutorul standardului IEC 61499 este necesar să se definească reguli referitoare la funcţionarea maşinilor cu stări (ECC). Aceasta se face prin intermediul unui mediu de execuţie care are rolul de a gestiona evenimentele, planificarea şi executarea funcţiilor bloc, precum şi transferul datelor între acestea. Planificarea executării funcţiilor bloc se poate realiza în mai multe moduri. Astfel, planificarea execuţiei în baza evenimentelor poate determina declanşarea unor noi evenimente, respectiv execuţia unor noi funcţii bloc. În situaţia în care există un număr de evenimente concurente, este necesară implementarea unui mecanism de cozi de execuţie. În acest caz comportamentul general al sistemului depinde de configuraţia cozii de execuţie. Spre exemplu, mediul de execuţie FBRT foloseşte o abordare de acest fel. O abordare alternativă constă în planificarea executării funcţiilor bloc în mod ciclic. În acest caz, fiecare funcţie bloc este executată în cadrul unui ciclu. Ţinând cont de aceste specificaţii ale standardului IEC 61499, în continuare se propune investigarea unei soluţii pentru executarea funcţiilor bloc în cadrul unor containere Docker, astfel:

- Implementarea unei aplicaţii care să gestioneze execuţia funcţiilor bloc într-un mod corespunzător cu evenimentele generate şi cu starea internă a unei funcţii bloc. Această aplicaţie reprezintă mediul de execuţie şi este similară cu instrumentul FBRT. Ca limbaj de programare se propune Python.

- Experimentarea utilizării unor algoritmi avansaţi de control ce sunt abstractizaţi sub forma unor funcţii bloc. Aceasta se poate face prin “împachetarea” unor fişiere executabile în cadrul unui obiect Python ce reprezintă o funcţie bloc.

- Fiecare funcţie bloc va fi implementată separat sub forma unui program Python. Aceasta se va executa în cadrul propriului virtualenv specific Python, respectiv se va crea un mediu de execuţie izolat.

- Pentru a fi conform cu specificaţiile IEC 61499, o funcţie bloc ar corespunde unui virtualenv Python, o resursă care poate conţine mai multe funcţii bloc ar corespunde unui container Docker, iar un dispozitiv care poate conţine mai multe resurse ar corespunde unui grup de containere asociate.

- Pentru interfaţa cu procesul sau interfaţa cu utilizatorul sunt necesare funcţii bloc de interfaţă (SIFB). Un exemplu ar fi pentru API-ul de tip REST. Acesta ar putea fi utilizat pentru interfaţa cu procesul, pentru a primi date de la un client, care poate fi la rândul lui un server OPC şi care primeşte date de la alţi clienţi OPC. Sau poate comanda unele dispozitive, unde se poate experimenta un exemplu cu ajutorul platformei Arduino.

- Interfaţa de tip OPC este necesară pentru comunicarea cu procesul controlat, cum ar fi instrumente, senzori sau elemente de execuţie, respectiv pentru a interfaţa sistemul de control cu lumea reală. Pentru comunicarea cu un sistem IEC 61499 este necesară o funcţie bloc de interfaţă (SIFB) care să implementeze standardul OPC. Aceasta ar putea să fie un server OPC. Dar acesta trebuie să fie executat de către mediul de execuţie IEC 61499. Similar şi în cazul interfeţei care implementează servicii web de tip REST, sau pentru interfaţa om-maşină (HMI), unde este necesar un GUI. Toate acestea ar trebui să fie SIFB-uri care sunt implementate şi executate conform cu IEC 61499.

- IEC 61499 a oferit o schemă XML pentru definirea sistemelor şi funcţiilor bloc. Similar, este nevoie de un format pentru a descrie structura sistemului, a funcţiilor bloc, precum şi a modului în care sunt acestea conectate. O soluţie în acest sens poate fi dată prin utilizarea limbajului YAML.

- Având sistemul descris într-un fişier YAML, respectiv funcţiile bloc şi modul cum sunt ele conectate, o aplicaţie va trebui să instanţieze aceste obiecte. Aceasta va utiliza API-ul Docker pentru a crea containere sau aplicaţii multi-container conform cu specificaţiile mediului de execuţie.

- Aplicaţia va avea un API care va permite utilizarea sa programatică, spre exemplu din cadrul unei interfeţe web, sau, în cazul unui API de tip REST, sub forma de serviciu web, realizând astfel un model de exploatare de genul Control-as-a-Service.

- Funcţia bloc este implementată sub forma unui program Python care poate executa la rândul său un alt executabil (algoritmul propriu-zis). În funcţie de modul cum a fost definit sistemul în fişierul YAML, se poate crea un container Docker care să conţină un virtualenv şi programul Python. Se poate însă să se creeze containere Docker cu mai multe virtualenv-uri, respectiv mai multe funcţii bloc sau aplicaţii cu mai multe containere care pot comunica în acelaşi segment de reţea. Pentru a comunica între containere este nevoie de funcţii bloc de tipul SIFB client-server (comunicare bidirecţională) sau publish-subscribe (comunicare unidirecţională).

- Izolarea unei funcţii bloc (program Python) se face la nivel de virtualenv, dar comunicarea între funcţiile bloc se face local şi nu necesită SIFB. Izolarea resurselor se face la nivel de container şi necesită SIFB pentru a putea comunica între ele.

Platforma World Wide Web (WWW) este foarte populară pentru programarea anumitor tipuri de aplicaţii distribuite cu interfeţe utilizator grafice (GUI). WWW a evoluat treptat şi neplanificat de la o platformă de livrare de documente la o arhitectură pentru programare distribuită. Adesea, o aplicaţie web trebuie să administreze un set de limbaje, de pildă, HTML, CCS şi JavaScript (pentru structurare, formatare de documente, respectiv, interactivitate) şi protocoale (de pildă, HTTP(S), pentru trimitere/primire de mesaje). Sunt folosite şi alte limbaje sau interfeţe de programare a aplicaţiilor (APIs), de pildă, SQL, pentru memorarea pe servere a datelor persistente şi structurate, sau JSON, pentru serializarea tipurilor de date complexe de transmis în reţea. În [8] se descrie un limbaj de programare funcţională, denumit Ur/Web, mai uşor de utilizat pentru programarea aplicaţiilor web moderne, reactive. Modelul adoptat de acest limbaj este unificat, iar programele scrise în acest limbaj pot fi compilate în alte limbaje standard pentru Web. Ur/Web suportă încapsularea stării specifice şi expune concurenţă simplă, pe baza unor tranzacţii, permiţând elaborarea unor aplicaţii distribuite, "multithreaded". Există implementări deschise ale acestui limbaj. Modelul de programare este simplu: un singur sistem distribuit cu un server controlat de programator şi mulţi clienţi (necontrolaţi de programator). Serverul şi clienţii comunică doar prin diferite canele de comunicaţie cu "tipuri puternice" (strongly typed), iar orice interacţiune apare ca parte a unei tranzacţii care se execută atomic, fără nici o interferenţă cu alte acţiuni executate în acelaşi timp. Serverul are acces la starea persistentă într-o bază de date SQL, care este de asemenea accesată doar prin canele cu tipuri puternice specifice schemei datelor. Clienţii îşi menţin interfeţele grafice printr-o variantă a programării funcţional-reactive. Limbajul Ur/Web oferă un model de programare unificat, dar expune explicit limbajele HTML şi SQL, însă fragmentele de cod în aceste limbaje sunt reprezentate ca valori cu tipuri puternice.

6.2 Arhitectură şi specificaţii cloud

Gestionarea sarcinilor se va realiza prin intermediul unei componente fixe (Service manager) care va asigura schimbul de informaţii cu aplicaţia web (Fig. 22). Interacţionarea utilizatorului cu această componentă se face prin intermediul acestei aplicaţii web. Această interfaţă transmite cererile utilizatorilor, utilizând servicii REST, către componenta de gestiune Service Manager. Aceasta va porni diferite instanţe şi va asigura execuţia algoritmilor sub forma unor regulatoare virtuale. Service manager va asigura următoarele:

- Preluarea datelor de configurare: ID algoritm/funcţie;
- Dacă utilizatorul a încărcat un fişier cu date de tip istoric, se va face analiza (parsing) acestui fişier şi salvarea datelor într-o anumită tabelă din baza de date;
- Controlul execuţiei funcţiilor: creare instanţe în conformitate cu configurarea din pagina web, pornire/oprire execuţie;
- Trimiterea unei confirmări către aplicaţia web în momentul în care s-a pornit sau s-a oprit execuţia unei funcţii;
- Afişarea stării comunicaţiei cu echipamentul din instalaţie, în cazul unei conexiuni de timp real.

Principalele sarcini ale componentei cloud sunt:

- Stocarea datelor de proces, în funcţie de identificatorii utilizatorului şi ai instalaţiei.

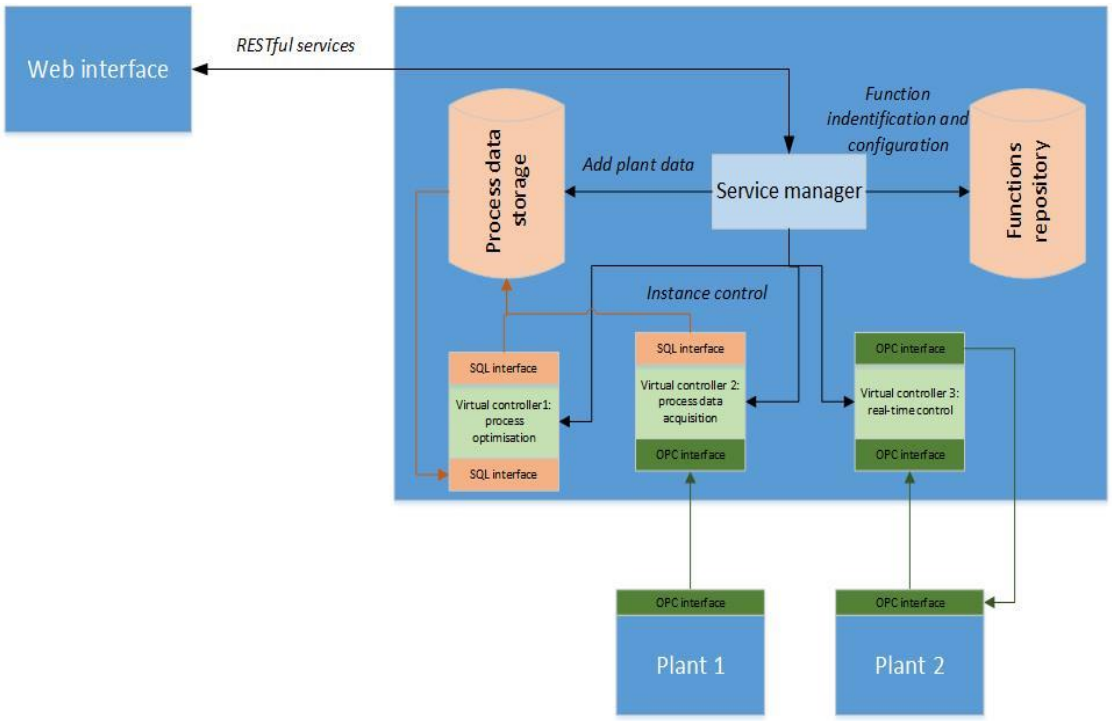

Figura 22: Gestionarea execuţiei funcţiilor (săgeţile negre reprezintă servicii RESTful, cele portocalii - funcţii de interfaţare SQL, iar cele verzi - date OPC).

- Gestionarea funcţiilor disponibile. Funcţiile vor fi stocate ca fişiere executabile, la care se ataşează interfeţele de comunicaţie prin OPC (pentru interfaţarea cu echipamentele din proces) sau SQL (pentru interfaţare cu baza de date). Considerând această structură modulară, ulterior interfeţele OPC ar putea fi înlocuite de alte interfeţe de comunicaţie de proces (de exemplu, publish/subscribe).
- Transferul de date între componenta Service manager şi baza de date sau componenta de execuţie a funcţiilor prin servicii RESTful.
- O componentă funcţie, cu interfeţe de date de intrare şi o interfaţă de date de ieşire poate fi considerată ca un regulator virtual. Pentru fiecare astfel de regulator se va aloca un container Docker în cadrul căruia să aibă loc execuţia. Este necesară deci gestionarea containerelor pe care se va face execuţia algoritmilor şi alocarea instanţelor algoritmilor pentru aceste containere.
- Gestionarea sarcinilor multiple de cereri de execuţie şi acces multiplu la baze de date.

Interfaţa web va avea rolul de gestionare a accesului utilizatorului la resursele bibliotecii (Fig. 23). Ea va permite:

- Configurarea detaliilor de conectare la instalaţie. Detaliile de configurare vor fi transmise utilizând servicii REST către Service manager.
- Posibilitatea de încărcare a unei serii de timp (fişier text sau xls), în scopul analizei offline.

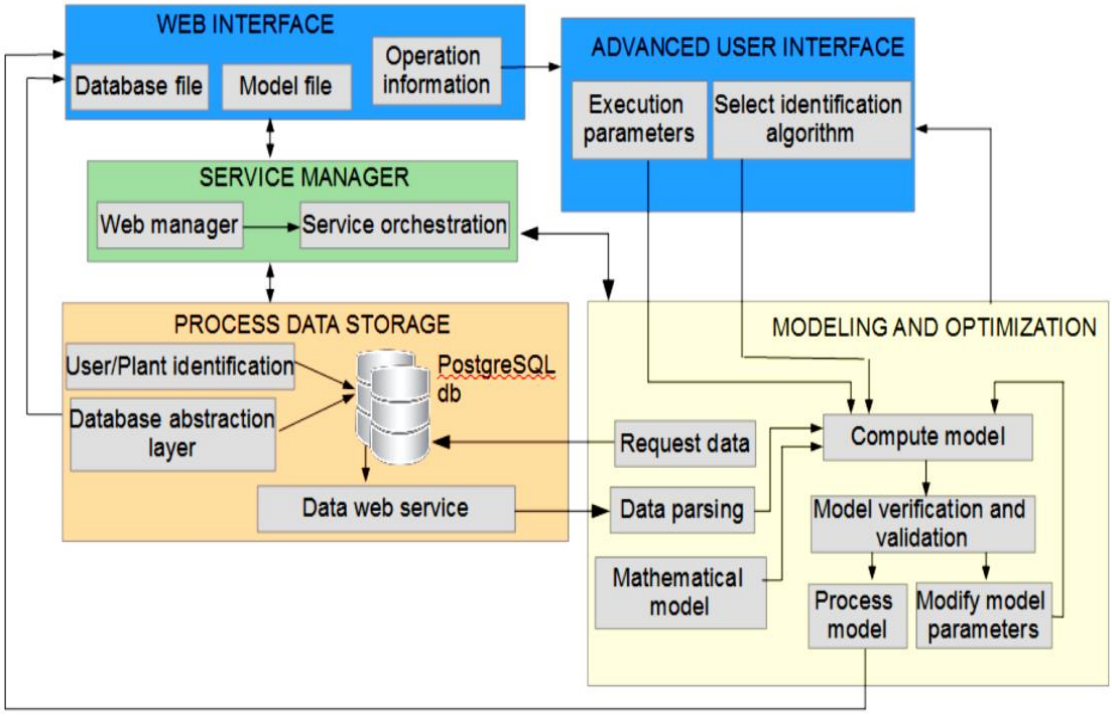

Figura 23: Gestionarea aplicaţiei web şi interconectarea cu baza de date de proces.

- Posibilitatea de urmărire a activităţii proprii a unui utilizator.
- Pornirea/oprirea execuţiei unei anumite funcţii.
- Posibilitatea de urmărire online a rezultatelor, dacă funcţia utilizează date de timp real.
- Posibilitatea de vizualizare a unui raport, dacă funcţia face o analiză pe un set de date de istoric.

Pentru îndeplinirea acestor funcţii este necesară dezvoltarea unei structuri SOA (Service Oriented Architecture), construită utilizând servicii Web, care să ofere o interfaţă pentru echipamentele industriale şi sistemele distribuite de control. Aceste sisteme de control vor fi abstractizate sub forma unor regulatoare virtuale şi interfaţate cu un API REST pentru a pune la dispoziţie un set de funcţii bloc de bază pentru constituirea serviciilor şi a algoritmilor complecşi. În acest mod, structurile de control de nivel înalt pot fi proiectate utilizând servicii standard şi flexibile, ce pot fi apoi implementate şi încărcate în cloud. Un model conceptual pentru serviciul Web de control este prezentat în Fig. 24.

Aplicaţiile de control se bazează pe conceptul de funcţii bloc, încapsulând datele şi algoritmii într-o structură de control abstractă. Aceste funcţii bloc sunt asociate echipamentelor controlate, dar ele vor putea fi executate de orice resursă de calcul disponibilă. De exemplu, mediul de execuţie pentru funcţiile bloc IEC 61499, FBDK, poate fi utilizat pe orice sistem care suportă JVM. S-a testat cu succes implementarea unei aplicaţii client-server utilizând FBDK prin încărcarea într-un container Docker. În acest mod, echipamentele ce formează un sistem de control distribuit pot fi pornite în interiorul propriului container şi pot fi interfaţate printr-o aplicaţie Java standard,

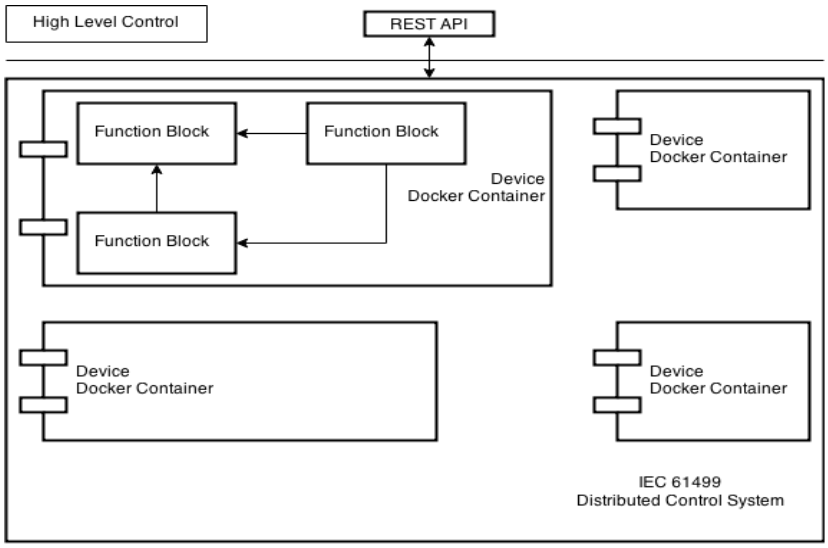

Figura 24: Model conceptual pentru serviciul Web de control.

implementată ca serviciu Web RESTful. Pe baza acestor servicii RESTful, pot fi construite atât structuri de control de nivel înalt, cât şi tipuri noi de modele, precum Control-as-a-Service (CaaS), în care algoritmi ce necesită o putere mare de prelucrare pot fi executaţi pe resurse virtuale, scalabile, create în funcţie de cerere.

6.3 Echilibrarea încărcării în cloud

Echilibrarea încărcării (Load Balancing) este o sarcină esenţială într-un mediu de calcul cloud, pentru a obţine utilizarea maximă a resurselor. Calculele în cloud pot avea loc într-un mediu fie static, fie dinamic, în funcţie de configurarea efectuată de furnizorul de resurse.

Mediul static. În mediul static, furnizorul de cloud instalează resurse omogene. În acest caz, resursele din cloud nu sunt flexibile. Este necesară cunoaşterea a priori a cerinţelor utilizatorului privind puterea de prelucrare, memoria, performanţa şi statisticile de utilizare [17].

Mediul dinamic. În mediul dinamic, furnizorul de cloud instalează resurse eterogene. În acest caz, resursele sunt flexibile. Procedurile nu se pot baza pe cunoaşterea apriorică, dar pot lua în considerare statisticile din timpul execuţiilor.

Echilibrarea încărcării pe baza distribuţiei spaţiale a nodurilor. Nodurile sunt distribuite pe scară largă în cloud. Nodul care ia decizia de repartizare a încărcării stabileşte, de asemenea, categoria de algoritmi de utilizat. Pot fi trei tipuri de algoritmi care decid care nod este responsabil pentru echilibrarea încărcării într-un mediu de calcul cloud.

Echilibrarea centralizată a încărcării. În tehnica centralizată de echilibrare a încărcării, toate deciziile de alocare şi planificare sunt făcute de un singur nod. Acest nod este responsabil pentru

stocarea bazei de cunoştinţe a întregii reţele din cloud şi poate aplica o procedură statică sau dinamică de echilibrare. Această tehnică reduce timpul cerut pentru analiza diferitelor resurse din cloud, dar creează o mare supraîncarcare în nodul central. De asemena, reţeaua nu este tolerantă la defecte/avarii, întrucât riscul căderii nodului central este mare, iar recuperarea după un astfel de eveniment poate fi dificilă.

Echilibrarea distribuită a încărcării. În tehnica de echilibrare distribuită a încărcării, asigurarea accesului de resurse şi deciziile de planificare a încărcării nu cad în sarcina unui singur nod. Nu există domeniu unic responsabil pentru monitorizarea reţelei din cloud, în schimb mai multe domenii monitorizează reţeaua pentru a lua o decizie corectă de echilibrare a încărcării. Fiecare nod din reţea menţine baze de cunoştinţe locale pentru a asigura o distribuţie eficientă a sarcinilor într-un mediu static şi redistribuirea acestora într-un mediu dinamic.

În scenariul distribuit, sunt compensate efectele defectării unui nod. Prin urmare, sistemul este tolerant la defecte şi echilibrat, deoarece nici un nod nu este supraîncărcat cu decizii de echilibrare a încărcării.

Echilibrarea ierahizată a încărcării. Echilibrarea ierarhizată a încărcării implică existenţa diferitelor niveluri ierarhice din cloud în luarea deciziei de echilibrare a încărcării. Astfel de tehnici de echilibrare a încărcării funcţionează mai ales în modul de lucru master-slave. Acestea pot fi modelate folosind structura de date arbore în care fiecare nod din arbore este echilibrat sub supravegherea nodului "părinte". Nodul master sau administrator poate utiliza un proces simplu pentru a obţine statistici ale nodurilor slave sau dependente ("copii"). Deciziile de asigurare a accesului sau de planificare se iau pe baza informaţiilor adunate de către nodul părinte.

Schema statică de echilibrare a încărcării permite cea mai simplă simulare şi monitorizare a mediului, dar nu reuşeşte să modeleze natura eterogenă a sistemului cloud. Pe de altă parte, algoritmii de echilibrare dinamică a încărcării sunt dificil de simulat, dar sunt cei mai potriviţi în mediul eterogen al prelucrării în cloud. În plus, nivelul nodului în care se implementează algoritmii statici sau dinamici, joacă un rol esenţial în stabilirea eficienţei lor. Spre deosebire de un algoritm centralizat, un algoritm distribuit oferă o mai bună toleranţă la erori, dar necesită un grad mai mare de replicare şi, pe de altă parte, un algoritm ierarhic distribuie încărcarea pe diferite niveluri ale ierarhiei, nodurile de nivel superior solicitând servicii de la nodurile de nivel inferior într-un mod echilibrat. Prin urmare, tehnicile dinamice de echilibrare a încărcării într-un mediu distribuit sau ierarhic oferă performanţe mai bune. Totuşi, performanţa mediilor de procesare în cloud poate fi maximizată pe viitor dacă dependenţele dintre sarcini sunt modelate folosind fluxuri de lucru.

Echilibrarea flexibilă a încărcării în cloud. Echilibrarea flexibilă a încărcării (EFI) este o soluţie de echilibrare a încărcării care îşi scalează (adaptează) automat capacitatea de manipulare a cererilor ca răspuns la traficul recepţionat. Procedura este folosită de exemplu în soluţiile de la Amazon Elastic Cloud (EC2) [18].

Numărul de accese simultane la o aplicaţie scalată în mod flexibil este utilizat pentru a determina numărul de instanţe necesare. Componentele unei aplicaţii distribuite trebuiesc scalate automat. Cererile trimise unei aplicaţii sunt folosite ca indicator pentru încărcarea cerută de volumul de calcul curent, pe baza căruia deduce apoi numărul necesar de instanţe componente, folosind numărului de cereri sincrone primite de mecanismul de echilibrare flexibilă şi alte informaţii de utilizare.

Unul dintre beneficiile majore ale echilibrării flexibile a încărcării este că reduce complexitatea gestionării, menţinerii şi scalării mecanismelor de echilibrare a încărcării. Acest serviciu este

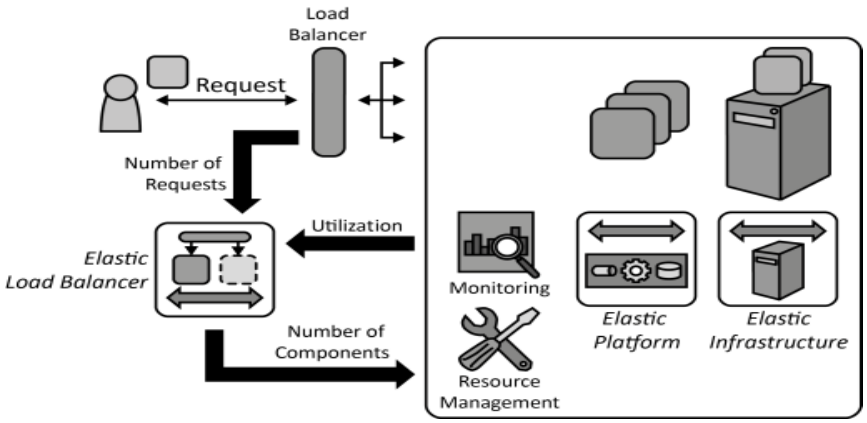

Figura 25: Arhitectura unui serviciu de echilibrare flexibilă a încărcării.

proiectat astfel încât să adauge sau să elimine în mod automat resurse în funcţie de necesităţi, fără a solicita nici o intervenţie manuală.

Există două componente logice ale arhitecturii unui serviciu de echilibrare flexibilă a încărcării: echilibratori de încărcare şi un serviciu de control. Echilibratorii de încărcare sunt resurse care monitorizează traficul şi gestionează cererile care vin prin intermediul Internetului. Serviciul de control monitorizează echilibratorii de încărcare, adaugă şi elimină o capacitate în funcţie de necesităţi, şi verifică dacă echilibratorii de încărcare funcţionează în mod corespunzator (Fig. 25).

Echilibrarea flexibilă a încărcării distribuie automat traficul de intrare între mai multe instanţe Amazon EC2. Acesta poate fi configurat ca un mecanism de echilibrare flexibilă a încărcării pentru a repartiza traficul recepţionat prin instanţe Amazon EC2 într-o singură Zonă de Disponibilitate sau în mai multe Zone de Disponibilitate. Echilibrarea flexibilă a încărcării permite atingerea unei şi mai mari toleranţe la erori pentru aplicaţii şi, în plus, obţinerea echilibrării dorite a încărcării resurselor. De asemenea, prin plasarea instanţelor Amazon EC2 în multe zone de disponibilitate se asigură toleranţa la defecte (fiabilitatea) sistemului. Atunci când instanţele de procesare sunt subordonate unui mecanism de echilibrare flexibilă a încărcării, se poate obţine o mai bună toleranţă la defecte deoarece acesta poate distribui traficul între mai multe zone de disponibilitate. În acest mod se garantează că numai instanţele EC2 valide vor recepţiona traficul.

Echilibrarea flexibilă a încărcării detectează de asemenea şi starea de sănătate a instanţelor EC2. Atunci când detectează instanţe Amazon EC2 nesănătoase, nu mai distribuie trafic pe rutele acestora. În schimb, traficul este direcţionat către instanţele EC2 sănătoase rămase. În plus, echilibrarea flexibilă a încărcării este în sine un sistem distribuit, care este tolerant la defecte şi monitorizat în mod activ.

Echilibrarea flexibilă a încărcării oferă de asemenea integrarea cu procesul de autoscalare, care asigură faptul că este disponibilă suficientă capacitate de procesare pentru a satisface diferitele niveluri (volume) de trafic.

6.4 Gestionarea incertitudinilor volumului de trafic pentru echilibrarea încărcării în cloud

Egalizarea încărcării este un element cheie pentru furnizarea de resurse suficiente pentru soluţii de prelucrare în cloud fiabile, iar performanţa acestor soluţii depinde de încărcarea dată de volumul de trafic. Se urmăreşte ca aceste resurse să ofere performanţe ridicate, disponibilitate ridicată, precum

şi soluţii sigure şi scalabile pentru a sprijini aplicaţiile găzduite. Multe aplicaţii mari consumatoare de conţinut prezintă un comportament de încărcare flexibil, care poate să crească până dincolo de puterea potenţială de procesare a unui singur server. În acelaşi timp, furnizorii de servicii au nevoie de o capacitate dinamică de resurse server pentru a implementa cât de multe servere au nevoie, în funcţie de necesităţi, într-un mod transparent. În cloud, structuri de servere încărcate echilibrat servesc drept un singur server virtual.

Abordarea de tip cloud echilibrat extinde modelul de implementare arhitectural de servere cu încărcare echilibrată tradiţionale, pentru a permite utilizarea lor împreună cu serverele de încărcare globale echilibrate din cloud [11]. Această abordare creşte cererea de opţiuni, care sunt asociate cu decizii bazate pe ambele obiective, tehnice şi de afaceri, ale desfăşurării aplicaţiilor de încărcare globale echilibrate de rutare. Pentru că multe studii empirice au arătat că distribuţia Pareto, tipică pentru traficul auto-similar (self-similar), oferă o potrivire bună pentru o serie de caracteristici importante ale reţelelor de comunicaţii şi servere Web, este justificat să se considere că mărimea cererilor utilizatorilor şi a pachetelor manipulate în cloud, satisfac distribuţii ne-Gaussiene, de tip "heavy-tailed".

Performanţa Internetului în sine depinde în mare parte de funcţionarea protocoalelor de rutare care îi stau la bază. Optimizarea rutării devine din ce în ce mai mult o provocare datorită naturii dinamice şi nesigure a traficului curent. Protocoalele de rutare utilizate în prezent în reţelele IP calculează configuraţii de rutare bazate pe topologia reţelei şi pe unele cunoştinţe aproximative asupra traficului (de exemplu, cel mai rău caz de trafic, traficul mediu, previziuni pe termen lung), fără a ţine cont de încărcarea curentă a traficului pe rutere şi legături, sau de posibile probleme de trafic, şi, prin urmare, nu acordă o atenţie deosebită problemei incertitudinii de trafic. Reţelele de mari dimensiuni posedă de obicei resurse în exces, iar modificările de rutare din cauza variaţiilor de trafic nu apar prea des. Cercetările efectuate în domeniul optimizării rutării consideră că abordarea de astăzi nu mai poate fi adecvată pentru a gestiona tiparele de trafic actuale, ci sunt necesare mecanisme de rutare mai fiabile pentru a efectua şi menţine funcţiile de reţea, în cazul defectării componentelor, în special în cazul apariţiei de evenimente neaşteptate de trafic, reprezentate de anomalii de volum.

Având în vedere o singură matrice cunoscută de trafic (Traffic Matrix — TM), optimizarea rutării constă în calcularea unui set de căi de origine-destinaţie şi o distribuţie a traficului între aceste căi, cu scopul de a optimiza unele criterii de performanţă, exprimate de obicei prin intermediul unei funcţii de cost. Abordări tradiţionale de optimizare a rutării au tratat problema bazându-se pe un mic grup de matrice TM reprezentative sau estimate, pentru a calcula configuraţii de rutare optime şi de încredere. Aceste tehnici menţin de obicei o istorie a matricelor TM observate în trecut, şi optimizează rutarea pentru traficul reprezentativ extras din matricele TM observate în timpul unei anumite ferestre de istorie. În acest sens, ne vom referi la algoritmi tradiţionali de rutare bazaţi pe predicţie. Cea mai bine cunoscută şi aplicată, în mod tradiţional, soluţie de rutare bazată pe predicţie este să se optimizeze rutarea pentru un scenariu de trafic de tipul celui mai rău caz posibil, folosind, de exemplu, matricea TM de la cea mai aglomerată oră găsită în istoria de trafic. Aceasta nu poate fi o soluţie rentabilă, dar operatorii s-au bazat în mod tradiţional pe supradimensionarea reţelelor lor pentru a suporta pierderea de performanţă preconizată. Soluţii mai bune sunt:

1. optimizarea rutării pentru o matrice TM estimată; această abordare poate oferi soluţii mai eficiente, dar depinde în mare măsură de calitatea tehnicii de estimare.

2. optimizarea rutării pentru mai multe matrice TM simultan, folosind, de exemplu, un număr finit de TM-uri dintr-o zi anterioară, din aceeaşi zi a săptămânii anterioare, etc.

În ambele abordări, trebuie să fie utilizate metode diferite pentru a găsi configuraţii de rutare cu o

medie bună şi cel mai rău caz de performanţă peste aceste TM-uri, cu observaţia că cel mai rău caz este numai pentru eşantioanele utilizate în realizarea optimizării, şi nu printre toate matricele TM posibile. Toate aceste abordări tradiţionale tind să funcţioneze destul de bine, dar au nevoie cu siguranţă de un mare efort de management, pentru a asigura robusteţea împotriva variaţiilor de trafic neaşteptate.

Pentru a face faţă atât creşterii dinamicii traficului cât şi incertitudinii şi nevoii obţinerii unor soluţii rentabile au apărut două noi abordări diferite: Rutarea Robustă (RR) [22] şi Echilibrarea Dinamică a încărcării (Dynamic Load-Balance — DCL) [15]. Abordarea RR tratează incertitudinea de trafic într-un mod de calcul off-line preventiv, anume, o configuraţie de rutare fixă este optimizată pentru un set mare de posibile solicitări de trafic. Dimpotrivă, echilibrarea dinamică a încărcării livrează trafic între mai multe căi fixe într-un mod reactiv on-line, de adaptare la variaţiile de trafic.

În rutarea robustă, incertitudinea de trafic este luată în considerare în mod direct la optimizarea rutării, calculându-se o singură configuraţie de rutare pentru toate cererile de trafic din cadrul unui anumit domeniu de incertitudine în care se presupune că traficul va varia. Acest domeniu de variaţie poate fi definit în moduri diferite, în funcţie de informaţiile disponibile: traficul ocupat pe ore, cele mai mari valori ale sarcinii link-urilor văzute anterior, un set de cerinţe de trafic observate anterior etc. Criteriul pentru a căuta această configuraţie unică de rutare este în general cel de reducere la minimum a utilizării maxime a legăturii pentru toate cerinţele domeniului de variaţie considerat. Estimarea precisă a fluxului de trafic de la origine la destinaţie (OD) oferă informaţii valoroase pentru sarcinile de management al reţelei. Cu toate acestea, lipsa de observaţii la nivel de desfăşurare a traficului, precum şi anomaliile intenţionate sau neintenţionate, reprezintă provocări majore pentru atingerea acestui obiectiv. Pentru a compensa intrinseca dimensionalitate scăzută a fluxurilor OD şi raritatea anomaliilor, un algoritm robust de echilibrare permite estimarea valorilor nominale şi anormale de trafic, folosind un subset mic de valori de trafic (eventual anormale), pe lângă numărul de legături, şi poate chiar determina cu precizie traficul nominal şi anomaliile rare în cazul unei matrice de trafic vagi. Rezultatele oferă date valoroase despre tipurile de măsurare şi scenarii de reţea care conduc apoi la estimarea mai precisă a traficului.

Echilibrarea Dinamică a încărcării (DLB) tratează situaţiile de incertitudine şi variaţie a traficului şi prin împărţirea traficului între mai multe căi on-line. În acest sistem dinamic, fiecare pereche origine-destinaţie de noduri din cadrul reţelei este conectată prin mai multe căi configurate anterior, iar problema este pur şi simplu de a distribui traficul între aceste căi, în scopul de a optimiza o anumită funcţie de cost. DLB este definită, în general, în ceea ce priveşte o funcţie legătură-cost, porţiunile de trafic fiind ajustate pe fiecare pereche origine-destinaţie de noduri, în scopul de a minimiza costul total pe reţea. Din motive de utilizare eficientă a resurselor, problema DLB are nevoie de mai multă atenţie în cazul procesării în cloud, din cauza naturii sale de procesare la cerere (On Demand Computing). Procesarea în cloud consideră ansamblul de resurse de calcul puse în comun şi reclamă distribuirea corectă a resurselor între sarcinile partajate, în caz contrar, în unele situaţii, anumite resurse pot fi supra- sau sub-utilizate. În abordarea de tip DLB, fiecare agent joacă un rol foarte important, fiind o entitate software definită de obicei ca un program independent care rulează sub controlul unui administrator de reţea, reduce costurile de comunicare a serverelor, şi accelerează rata de echilibrare a încărcării, îmbunătăţind astfel în mod indirect productivitatea şi timpul de răspuns al sistemului cloud.

O analiză comparativă între algoritmii de echilibrarea a încărcării robuşti şi cei dinamici conduce la următoarele concluzii. Algoritmii care promovează mecanisme de tip DLB subliniază, printre altele, faptul că această echilibrare reprezintă soluţia de utilizare cea mai eficientă a resurselor, adaptându-se la încărcarea reţelei într-un mod automat şi descentralizat. Pe de altă parte, cei care susţin utilizarea RR, spun că nu este de fapt nevoie să se pună în aplicare mecanisme dinamice de rutare complicate, şi că pierderea de performanţă suportată pentru utilizarea unei singure configuraţii

de rutare este neglijabilă în comparaţie cu creşterea complexităţii. O caracteristică interesantă a RR se bazează pe utilizarea unei singure configuraţii fixe de rutare, evitându-se posibilele instabilităţi datorate modificărilor de rutare. În practică, operatorii de reţele sunt reticenţi în a utiliza mecanisme dinamice şi fixe şi preferă configuraţii de rutare fixe, considerând ca aşa au mai multe informaţii relativ la ceea ce se întâmplă în reţea. Primul neajuns pe care îl identificăm într-o paradigmă de rutare robustă îl reprezintă criteriul cost-eficienţă asociat. Folosirea unei singure configuraţii de rutare pentru perioade lungi de timp poate fi extrem de ineficientă. Definiţia incertitudinii stabilită în RR stabileşte un compromis critic între performanţă şi robusteţe: seturi mai mari permit să se ocupe de un grup mai larg de cereri de trafic, dar cu preţul unei rutări ineficiente; pe de altă parte, seturi mai restrânse produc scheme mai eficiente de rutare, dar sub rezerva unor garanţii slabe de performanţă.

În cazul anomaliilor de volum, o extensie dinamică a RR, cunoscută sub numele de Rutare Robustă Reactivă (Reactive Robust Routing — RRR), permite depăşirea acestui neajuns [13]. Abordarea RRR utilizează metoda detecţiei/localizării secvenţiale a volumelor anormale de trafic pentru a detecta şi a localiza rapid schimbările bruşte în cererile de trafic şi decide apoi modificări de rutare. O abordare de tip echilibrare a încărcării pentru RRR, în care traficul este distribuit între căi fixate, conform unei entităţi centralizate care controlează fracţiunile de trafic trimise pe fiecare cale, poate rezolva această problemă.

Al doilea dezavantaj pe care îl identificăm în RR este legat de funcţia obiectiv pe care intenţionează să o minimizeze. Optimizarea în condiţii de incertitudine este, în general, mult mai complexă decât optimizarea clasică, forţată sa folosească nişte criterii de optimizare simple, cum ar fi utilizarea maximă a legăturii (Maximum Link Utilization — MLU), adică minimizarea încărcării pe legătura cea mai folosită din reţea. MLU este de departe cea mai populară funcţie obiectiv pentru evaluarea traficului (Traffic Engineering), dar în mod clar nu este cel mai potrivit criteriu de optimizare la nivel de reţea; concentrarea prea mare pe MLU conduce adesea la distribuţii mai rele de trafic, care afectează în mod negativ încărcarea medie a reţelei şi, astfel, întârzierea totală de la un capăt la altul (end-to-end) din reţea, care reprezintă un indicator important al QoS [34]. Este uşor de observat că minimizarea MLU într-o topologie de reţea cu legături eterogene poate conduce la rezultate slabe în ceea ce priveşte performanţa globală a reţelei. Pentru a evita această problemă, se poate minimiza utilizarea medie a legăturii în loc de MLU. Utilizarea medie a legăturii oferă o imagine mai bună asupra performanţei la nivel de reţea, deoarece nu depinde de o anumită sarcină sau de capacitatea fiecărei legături dintr-o reţea, ci de valoarea medie. În ciuda acestui avantaj, o minimizare directă a utilizării medii a legăturii nu garantează un anumit MLU, deci este nevoie să impunem o limită de utilizare pentru MLU.

În DLB, traficul este împărţit între căi fixe, stabilite a priori, cu scopul de a optimiza o anumită funcţie de cost. De exemplu, o funcţie de cost pe legătura stabilită pe baza măsurătorilor întârzierii poate duce la o mai bună performanţă la nivel global din perspectiva QoS. DLB are un avantaj, anume acela de a menţine rutarea adaptată la traficul dinamic. Cu toate acestea, algoritmii DLB reprezintă un compromis între adaptabilitate şi stabilitate, care ar putea fi deosebit de dificil de menţinut în cazul unor schimbări semnificative şi bruşte de trafic.

Comparând RR şi DLB, cea mai bună metodă pare a fi o abordare dinamică, care constă în calcularea unei rutări optime pentru fiecare cerere de trafic i şi evaluarea performanţei ei pentru următoarea cerere de trafic $i+1$, pe baza unei serii de timp date de cereri de trafic. Această abordare dinamică permite evitarea a două neajunsuri importante ale DLB:

- adaptarea în DLB este iterativă şi niciodată instantanee;
- în toate mecanismele DLB, căile sunt stabilite a priori şi rămân neschimbate în timpul funcţionării, deoarece fiecare nouă optimizare de rutare poate schimba nu numai porţiuni de trafic,

dar chiar şi căile însele.

O analiză cuprinzătoare a diferitelor soluţii posibile pentru această problemă, între care tradiţionala Optimizare a Rutării pe Baza Predicţiei, Optimizarea bazată pe Rutarea Robustă şi Echilibrarea Dinamică a încărcării, a condus la următoarele concluzii:

1. Atunci când traficul este relativ dinamic, nu este rentabilă folosirea unei singure configuraţii de rutare. Metoda tradiţională de Optimizare a Rutării pe Baza Predicţiei Tradiţionale poate conduce la configuraţii destul de ineficiente, sau chiar irealizabile de rutare, atunci când traficul este nesigur şi dificil de prognozat. Optimizarea bazată pe Rutarea Robustă oferă garanţii de performanţă în pofida incertitudinii de trafic, dar compromisul asociat între robusteţea şi eficienţa de rutare poate fi deosebit de dificil de gestionat cu o singură configuraţie de rutare.

2. Este evident faptul că este necesară o anumită formă de dinamism, fie sub formă de Rutare Reactivă Robustă şi Echilibrare a încărcării (RRLB) sau Echilibrare Dinamică a încărcării (DLB). RRLB calculează o configuraţie de rutare robustă nominală şi oferă o configuraţie de rutare robustă alternativă (folosind aceleaşi căi din timpul funcţionării normale) pentru orice situaţie atipică posibilă de flux OD. Pentru a detecta aceste anomalii, măsurătorile de încărcare a legăturilor trebuie colectate şi prelucrate de către o entitate centrală. Pe de altă parte, DLB colectează aceleaşi măsurători, dar necesită, de asemenea, actualizarea echilibrării încărcării într-un timp relativ scurt. Complexitatea adăugată este apoi de a distribui aceste rezultate către toate ruterele şi de a actualiza on-line distribuirea încărcării.

3. Complexitatea suplimentară implicată de DCL nu este justificată atunci când traficul nu variază semnificativ. În cazul unor anomalii mari de volum, algoritmii DLB furnizează, în general, rezultate mai bune decât RRLB după convergenţă, dar ei prezintă un comportament tranzitoriu nedorit. Pe de altă parte, algoritmii RRLB nu suferă, în principiu, de această problemă, deoarece fracţiunile de echilibrare a încărcării sunt calculate off-line în mod robust, profitând de avantajul oferit de paradigma de rutare robustă.

4. Utilizarea unor algoritmi DLB, devine foarte atrăgătoare atunci când anomaliile de volum sunt dificil de localizat. Comportamentul tranzitoriu pe care aceştia îl prezintă la modificări mari de trafic poate fi controlat în mod eficient, sau cel puţin atenuat, prin mecanisme simple, ceea ce duce la un grad mai mare de utilizare maximă a legăturii şi, în general, la o performanţă mai bună la nivel global.

5. Un criteriu de performanţă locală, cum ar fi utilizarea maximă a legăturii, nu reprezintă o funcţie obiectiv adecvată în ceea ce priveşte performanţa reţelei globale şi asigurarea QoS. Utilizarea maximă a legăturii este folosită pe scară largă în actualele probleme de optimizare de reţea, prin urmare, acest lucru ar trebui să fie luat în considerare pentru îmbunătăţirea implementărilor viitoare.

6. Utilizând o combinaţie simplă de indicatori de performanţă, cum ar fi utilizarea maximă şi medie a legăturii, se poate obţine o configuraţie robustă de rutare care surclasează cu siguranţa implementările actuale dintr-o perspectivă globală “end-to-end”, producând în acelaşi timp rezultate foarte similare pentru cel mai rău caz de utilizare a legăturii. De fapt, aceasta înseamnă că funcţiile obiectiv de optimizare pot fi păstrate simple, permiţând totuşi obţinerea unei performanţe mai bune la nivel de reţea.

7. Un dezavantaj major al RR este dependenţa inerentă de definirea domeniului de incertitudine: intervale mai mari permit tratarea unui grup mai larg de cereri de trafic, dar cu preţul unei

rutări ineficiente; pe de altă parte, intervale mai restrânse produc scheme mai eficiente de rutare, dar sub rezerva unor garanţii slabe de performanţă.

6.5 Mecanisme de acces în IoT şi Cloud

Scopul urmărit a fost de a integra arhitectura bazată pe sensibilitate la context, propusă în aplicaţiile bazate pe IoT, ca platformă eficientă de programe bazată pe servicii folosind o abordare middleware, numită Platforma de Control sensibil la Context (Context-aware Control Platform — CaCP). Platforma separă nivelul de achiziţie de codul aplicaţiilor şi se ocupă de problemele de gestionare a datelor de context. Pe post de componentă middleware, platforma oferă mai multe servicii pentru aplicaţii, cum ar fi procesarea interogărilor, gestionarea datelor de la senzori, monitorizarea reţelei de senzori, prelucrarea informaţiilor sensibile la context, etc. Rolul său este de a pune la dispoziţia utilizatorilor de aplicaţii datele achiziţionate de la reţele de senzori diverşi şi de a furniza servicii Web. Prin urmare, resursele middleware pot fi accesate prin aplicaţii, prin intermediul interfeţelor de servicii Web.

Principalul avantaj al utilizării acestei arhitecturi stratificate este că permite reconfigurarea şi extensibilitatea fără a modifica aplicaţiile. Cu toate acestea, îndeplinirea cerinţelor stricte în timp real este dificil de realizat în arhitecturi stratificate, deoarece fluxurile de date traversează mai multe niveluri înainte de a ajunge la nivelul aplicaţiei. Prin urmare, arhitectura sensibilă la context propusă este mai potrivită cerinţelor mai relaxate de operare în timp real, deoarece poate tolera latenţe mai mari de timp şi poate funcţiona în continuare în timp ce se adaptează la schimbările de mediu.

6.5.1 Modelare de acces IoT

Primul obiectiv în utilizarea CaCP este de a realiza accesul într-un mediul IoT, ca o entitate care poate interacţiona atât cu domeniul IoT cît şi cu mediul ambiant. Această entitate reprezintă un “obiect” în Internetul Obiectelor (IoT), şi este principalul obiectiv al interacţiunilor dintre agenţi software. Acest obiect este reprezentat de două componente: hardware şi software. Componenta hardware a entităţii este denumită “dispozitiv”. Dispozitivul conectează entitatea la mediul său pentru a-l monitoriza. Componenta software, care oferă informaţii cu privire la entitate şi permite controlul dispozitivului, este o “resursă”. Exploatarea resurselor depinde de suportul hardware real al dispozitivului, astfel încât este necesar un “serviciu” pentru a oferi toate funcţionalităţile de a interacţiona cu alte entităţi şi procese conexe, prin intermediul unei interfeţe standardizate. Serviciile asigură funcţionalitatea unui dispozitiv prin accesarea resurselor sale. Relaţiile dintre entităţi şi servicii sunt denumite “asociaţii”.

Figura 26 ilustrează relaţiile dintre cele patru elemente ale unui model de acces IoT. Să observăm că serviciile furnizate de modelul de acces din Fig. 26 corespund categoriei Semantic Web Services. În conformitate cu specificaţiile OWL, resursele de Web Semantic furnizează servicii care sunt descrise de profilul serviciului, modelul serviciului şi descrierea serviciului [21]. Un profil de serviciu trebuie să conţină informaţii despre entitatea căreia îi este asociat şi legătura cu resursa care oferă acest serviciu. Profilul serviciului descrie un serviciu pe baza intrărilor, ieşirilor, condiţiilor prealabile şi efectelor. Un serviciu care are nevoie ca o intrare să fie procesată de către o resursă, formulează această nevoie ca pe o proprietate. În conformitate cu această proprietate, serviciul este legat de o entitate. Prin acţiunile lor, serviciile schimbă proprietăţile entităţilor dintr-o stare iniţială (specificată ca precondiţie) la o stare dorită (specificată ca efect).

Utilizatorii IoT au acces la resurse din IoT prin intermediul unei interfeţe de acces, care se adresează unui Serviciu Web. Detaliile tehnice necesare utilizatorilor pentru a accesa serviciul sunt specificate în descrierea serviciului, care precizează legatura (maparea) dintre entitatea specifică

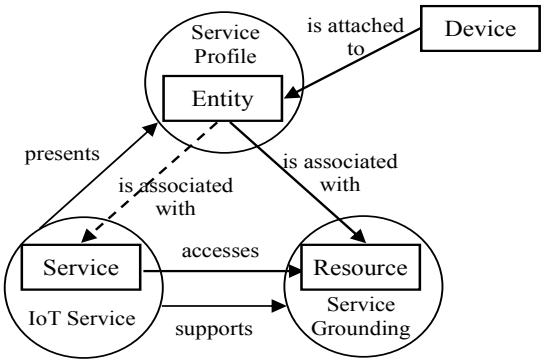

Figura 26: Interacţiuni ale componentelor într-un model de bază de acces IoT.

din domeniu şi atributele proprietăţilor observabile de senzori. Fiecare atribut al entităţii alocate în profilul serviciului este atribuit unei anumite proprietăţi (de exemplu, un anumit tip de măsurare).

6.5.2 Modelare de acces cloud

Valorificarea avantajelor oferite de utilizarea Internetului în efectuarea unor servicii eficiente ar fi incompletă în cazul în care nu s-ar folosi avantajele oferite de tehnologia cloud. Dezvoltarea IoT aduce de asemenea multe aplicaţii potentiale. Tehnologia cloud permite procesarea datelor pe scară largă şi îmbunătăţeşte flexibilitatea ca rezultat al creşterii numărului de reţele conectate de dispozitive IoT şi, prin urmare, a cantităţii mari de informaţii care trebuie prelucrate. Se urmăreşte facilitarea accesului în IoT a reţelelor de senzori, astfel încât să transforme infrastructura existentă de detecţie în conformitate cu o paradigmă IoT centrată pe prelucrarea în cloud (cloud-centric). În cazul particular de integrare a unei platforme middleware sensibile la context, componentele sale pot fi distribuite în diferite niveluri de abstractizare din cloud, corespunzător tipului de serviciu realizat, aşa cum este ilustrat în Fig. 27, în care sunt prezentate de asemenea şi celelalte componente hardware şi software ale unei aplicaţii de control sensibile la context.

1. Nivelul de Servicii pentru senzori (Sensor as a Service — S^2aaS) permite accesul la componentele hardware (senzori individuali, noduri de reţele de senzori, etichete RFID), printr-o interfaţă de acces a reţelei senzorului. Senzorii fizici şi interfaţa de comunicare sunt livrate de către furnizorii de senzori.

2. Nivelul de servicii de Infrastructură (Infrastructure as a Service — IaaS) oferă utilizatorilor infrastructurii acces la infrastructura cloud, incluzând reţeaua, servere, sisteme de operare, stocare. Consumatorii pot astfel implementa şi rula un software oarecare pentru aplicaţii specifice.

3. Nivelul de serviciu Platformă (Platform as a Service — PaaS) permite utilizatorilor să implementeze pe infrastructura cloud aplicaţiile achiziţionate sau nou create, cu ajutorul unor limbaje de programare familiare, biblioteci, servicii şi instrumente oferite de către furnizor. Componenta software a platformei middleware sensibile la context este plasată la acest nivel.

4. Nivelul de servicii Software (Software as a Service — SaaS) furnizează toate programele necesare pentru aplicaţii, inclusiv cele de mesagerie, management, dezvoltare, şi aşa mai departe. Acesta conţine, de asemenea, interfeţe dedicate pentru interacţiunea cu operatorii umani.

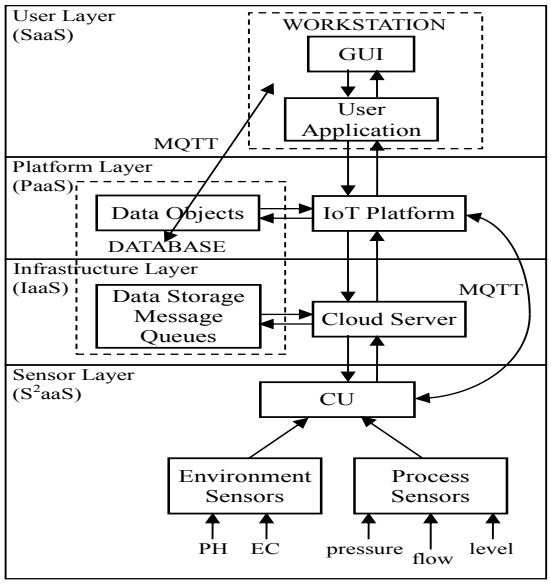

Figura 27: Distribuţia unei structuri IoT asociate cu middleware sensibil la context.

6.6 Aplicaţie în timp real pe platforma Cloud ICIPRO

Sistemul de operare CentOS (disponibil pe Platforma Cloud ICIPRO) permite rularea unor aplicaţii de timp real. CentOS (Community ENTerprise Operating System) este un sistem de operare GNU/Linux, liber şi gratuit, bazat pe Red Hat Enterprise Linux, de la firma Red Hat. Acest sistem de operare a fost dezvoltat pentru a oferi o distribuţie gratuită pentru anumite domenii de activitate. CentOS este cea mai populară distribuţie GNU/Linux pentru servere Web, deţinând o cotă de utilizare de circa 30% dintre serverele Web existente.

CentOS aduce noutăţi importante, cum ar fi: suportul pentru sistemul de fişiere ext4, posibilitatea folosirii noilor discuri rapide SSD, îmbunătăţiri privind consumul de energie în centrul de date, mecanisme de securitate mai rafinate (securitatea informatică Red Hat fiind certificată internaţional la nivelul exigenţelor militare), noi interfeţe grafice (KDE 4.3), precum şi progrese masive în tehnologiile de virtualizare (la nivel de server şi la nivel de desktop). Compania Red Hat este unul dintre dezvoltatorii importanţi în virtualizare, mai ales pentru centre mari de date şi servere pentru aplicaţii critice.

Distribuţia CentOS oferă variante pentru platforme Cloud computing compatibile IBM-PC (Intel/AMD), în arhitecturi de 32 şi 64 de biţi. CentOS este forma de GNU/Linux aleasă pentru cele mai multe servere Web ce se găsesc acum online; aceasta îl face să fie o distribuţie foarte populară, cu documentaţie bogată şi suport uşor de procurat. Din cauza similitudinii software-ului, documentaţia oficială Red Hat, disponibilă gratuit la adresa `http://docs.redhat.com/`, este aplicabilă şi pentru versiunea corespunzătoare de CentOS.

Nucleul CentOS include multitasking real, memorie virtuală, biblioteci partajate, încarcare de module la cerere (demand loading), executabile partajate, gestiunea optimă a memoriei şi reţele TCP/IP. CentOS este un nucleu monolitic cu încărcare de module. Driverele pentru dispozitive periferice şi extensiile de nucleu rulează în mod normal în inelul de bază al arhitecturii, cu acces total la hardware, deşi unele rulează în spaţiul utilizator. Spre deosebire de nucleele monolitice standard, driverele dispozitivelor se configurează uşor ca module şi se încarcă sau se descarcă în timpul rulării sistemului. De asemenea, ele dispun de posibilitatea de preempţiune (pot fi pre-empted) în anumite

condiţii. Această caracteristică de timp real a fost adăugată pentru a trata întreruperile hardware corect, şi pentru a îmbunătăţi suportul pentru multiprocesare simetrică. Preempţiunea îmbunătăţeşte latenţa, crescând viteza de răspuns şi făcând CentOS potrivit pentru aplicaţii de timp real. Dintre facilităţile oferite de sistemul de operare CentOS menţionăm semafoarele şi memoriile partajate, folosite în aplicaţiile în timp real. Aplicaţiile în timp real se bazează pe diverse mecanisme de control a concurenţei, oferite de sistemul de operare.

Semafor. Conceptul de semafor a fost introdus de cercetătorul olandez Edsger Wybe Dijkstra, pentru a facilita sincronizarea proceselor, prin protejarea secţiunilor critice şi asigurarea accesului exclusiv la resursele pe care procesele le utilizează. Un *semafor* reprezintă o variabilă partajată, care permite sau interzice accesul unui proces la o resursă comună. Secţiunile critice sunt secvenţe de instrucţiuni care pot genera erori sau conflicte şi a căror execuţie de către mai multe procese trebuie controlată.

În CentOS, noţiunea de semafor a fost generalizată prin posibilitatea de a executa mai multe operaţii simultan asupra unui obiect, fără a prejudicia rezultatul real. Astfel, se poate lucra cu mulţimi de semafoare care sunt descrise de structuri de date ce conţin o altă structură reprezentând permisiunile proceselor la semafoare, un pointer la primul semafor din set, momentul de timp la care a avut loc ultima operaţie asupra setului de semafoare. Accesul unui proces la un set de semafoare este controlat, deoarece doar procesele cu anumite caracteristici au dreptul să modifice semafoarele.

Monitor. Un *monitor* este o construcţie specială de timp real, cu un tip de date abstract. Scopul său principal este de a încapsula variabilele partajate şi operaţiile asupra acestora. Astfel, toate secţiunile critice sunt concentrate în această structură la care, la un moment dat, are acces unul singur dintre task-uri. Secţiunile critice sunt extrase din task-urile obişnuite şi devin proceduri sau funcţii ale monitorului.

Memorii partajate. *Memoria partajată* reprezintă un spaţiu comun de memorie, care poate fi accesat simultan de mai multe procese (taskuri) din sistem. Procesele din sistem pot comunica prin datele memorate în spaţiul unic de memorie partajată. Utilizarea unui spaţiu unic de memorie poate conduce la conflicte de acces la memorie, în lipsa unor mecanisme de control, atunci când mai multe procese încearcă să utilizeze aceeaşi zonă de memorie sau când doresc să utilizeze o variabilă partajată la care un alt proces are acces exclusiv.

Pentru a preveni situaţii de conflict este necesar ca actualizarea valorii unei variabile partajate să se realizeze ca o operaţie atomică, care nu poate fi întreruptă de către un alt proces, care doreşte să acceseze aceeaşi zonă de memorie partajată. Cel de-al doilea proces va putea actualiza variabila globală asociată zonei de memorie partajată, doar după ce primul proces a eliberat zona respectivă de memorie. Serializarea accesului la variabilele partajate se obţine prin implementarea mecanismelor de excludere mutuală. Pentru actualizarea în siguranţă a unei variabile globale este necesar ca operaţia respectivă să se execute într-o secţiune critică. Procesul care intră în secţiunea critică are dreptul exclusiv de a accesa variabila partajată asociată secţiunii respective. Atât timp cât procesul execută secţiunea critică, toate celelalte procese concurente nu vor avea dreptul să acceseze variabila partajată respectivă.

Aplicaţie. În cadrul proiectului a fost realizat un sistem compus din mai multe procese, care comunică prin intermediul unei zone de memorie partajată. Pentru asigurarea consistenţei datelor, zona de memorie partajată a fost protejată de un sistem de semafoare.

Primul proces modelează un Sistem industrial neliniar, cu timp mort. Acest proces are nişte mărimi de intrare (U) şi nişte mărimi de ieşire (Y), mărimi stocate în zona de memorie partajată. Procesul citeşte în buclă datele de intrare (U), calculează datele de ieşire (Y) şi le depune în câmpuri separate ale zonei de memorie partajată.

Un al doilea proces reprezintă Regulatorul, care citeşte în buclă, din memoria partajată, datele de la un sumator (E), calculează nişte mărimi de comandă şi le stochează în zona de memorie partajată, care reprezintă datele de intrare ale primului proces (U).

Al treilea proces este un Sumator, care citeşte (din zona de memorie partajată) datele de la procesul care stabileşte valoarea mărimii de Referinţă (R), face diferenţa cu datele de ieşire de la primul proces (Y) şi le stochează ca mărimi de intrare pentru procesul Regulator.

Procesul Referinţă preia datele de la un operator sau simulează nişte valori pentru mărimile de referinţă şi le memorează în zona de memorie partajată.

Această aplicaţie modelează comportarea unui sistem de reglare. Aplicaţia dispune de posibilitatea ca mărimile de referinţă să fie introduse de pe un alt calculator, aflat la distanţă. Pentru ca datele să fie transmise în timp real, comunicaţia se realizează prin intermediul unor socket-uri. În plus, aplicaţia dispune de două procese, care pot afişa numeric sau grafic evoluţia mărimilor din proces, acestea având posibilitatea să acceseze zona de memorie partajată.

7 Concluzii

Prezentarea făcută în acest raport dovedeşte că obiectivele propuse pentru această etapă a proiectului, Etapa III, au fost atinse. Toate cele cinci activităţi prevăzute au fost efectuate. Investigaţiile întreprinse sunt foarte utile pentru realizarea etapei finale.

Implementarea aplicaţiilor de conducere automată avansată a proceselor industriale necesită uzual resurse sporite de stocare şi execuţie. Raportul a prezentat o aplicaţie bazată pe cloud care îi permite unui utilizator să aibă acces la diferite strategii de conducere folosind o interfaţă web şi să beneficieze de regulatoare virtualizate, care să execute acei algoritmi şi să trimită comenzi dispozitivelor din proces. Raportul a definit cum modulele implementând aceste noi servicii pot fi interconectate. Abordarea inovativă prezentată presupune virtualizarea unor "baze de date" de regulatoare de proces şi îi conferă inginerului automatist dintr-o întreprindere industrială accesul la strategii avansate de modelare, optimizare şi conducere, implementate ca funcţii bloc IEC 61499.

Bibliografie

[1] V. Aggarwal, M. Mao, and U. M. O'Reilly. A self-tuning analog proportional-integral-derivative (PID) controller. In *First NASA/ESA Conference on Adaptive Hardware and Systems (AHS'06)*, pages 12–19, June 2006.

[2] Q. Bay. Analysis of particle swarm optimization algorithm. *Computer and Information Science*, 3(1):180–184, 2010.

[3] P. Benner, V. Sima, and M. Voigt. Algorithm 961: Fortran 77 subroutines for the solution of skew-Hamiltonian/Hamiltonian eigenproblems. *ACM Transactions on Mathematical Software (TOMS)*, 42(3):1–26, June 2016.

[4] C. Bischof, P. Khademi, A. Mauer, and A. Carle. Adifor 2.0: Automatic differentiation of Fortran 77 programs. *IEEE Computational Science and Engineering*, 3(3):18–32, 1996.

[5] M. E. Campbell and N. R. Ahmed. Distributed data fusion: Neighbors, rumors, and the art of collective knowledge. *IEEE Control Syst. Mag.*, 36(4):83–109, 2016.

[6] A. O. Castro, U. H. Bezerra, J. C. Leite, and M. S. S. Azevedo. Methodology proposal for multicriteria optimization using NSGA-II in industrial applications. In *2014 11th IEEE/IAS International Conference on Industry Applications*, pages 1–8, Dec 2014.

[7] O. Chenaru, A. Stanciu, D. Popescu, G. Florea, V. Sima, and R. Dobrescu. Modeling complex industrial systems using cloud services. In *Proceedings of The 20th International Conference on Control Systems and Computer Science (CSCS20)*, May 27–29, 2015, Bucharest, Romania, pages 565–571, 2015.

[8] A. Chlipala. Ur/Web: A simple model for programming the Web. *Communications of the ACM*, 59(8):93–100, 2016.

[9] M. Debruyne and T. Verdonck. Robust kernel principal component analysis and classification. *Advances in Data Analysis and Classification*, 4(2-2):151–167, 2010.

[10] T. Dumitrescu, R. Popescu, and R. Dobrescu. Design of an intelligent system for disasters management. In *Proceedings of the WSEAS International Conference on Mathematical Models and Computational Methods (MMMAS 2015)*, October 17–19, 2015, Agios Nikolaos, Crete, Greece, pages 128–133, 2015.

[11] A. Fayoumi. Performance evaluation of a cloud based load balancer severing Pareto traffic. *Journal of Theoretical and Applied Information Technology*, 32(1):28–34, 2011.

[12] Z.-L. Gaing. A particle swarm optimization approach for optimum design of PID controller in AVR system. *IEEE Transactions on Energy Conversion*, 19(2):384–391, June 2004.

[13] H. Gao, J. Liu, Y. Liu, Z. Wei, S. Huang, F. Zhang, and X. Li. A robust model for multistage distribution network planning. In *2015 International Symposium on Smart Electric Distribution Systems and Technologies (EDST)*, pages 37–41, Sept 2015.

[14] H. Gou, Y. Jing, and Y. Zhao. Query interface classification based on singular value decomposition. In *8th International Symposium on Computational Intelligence and Design (ISCID)*, 12–13 Dec., volume 1, pages 572–575. IEEE, 2015.

[15] J. Grover and S. Katiyar. Agent based dynamic load balancing in cloud computing. In *2013 International Conference on Human Computer Interactions (ICHCI)*, pages 1–6, Aug 2013.

[16] H. M. Hanisch, M. Hirsch, D. Missal, S. Preube, and C. Gerber. One decade of IEC 61499 modeling and verification — results and open issues. In *13th IFAC Symposium on Information Control Problems in Manufacturing*, Moscow, Russia, volume 42, pages 211–216, 2009.

[17] N. Haryani and D. Jagli. Dynamic method for load balancing in cloud computing. *IOSR Journal of Computer Engineering*, 16(4):23–28, 2014.

[18] A. Jagga. Amazon elastic cloud — the complete reference. *International Journal of Technology Enhancements and Emerging Engineering Research*, 1(1):1–4, 2013.

[19] J. S. R. Jang. ANFIS: Adaptive-network-based fuzzy inference system. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(3):665–685, May 1993.

[20] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings IEEE International Conference on Neural Networks*, Perth, Australia, volume IV, pages 1942–1948, 1995.

[21] J. Kiljander, A. D'elia, F. Morandi, P. Hyttinen, J. Takalo-Mattila, A. Ylisaukko-Oja, J. P. Soininen, and T. S. Cinotti. Semantic interoperability architecture for pervasive computing and Internet of Things. *IEEE Access*, 2:856–873, 2014.

[22] M. Mardani and G. B. Giannakis. Robust network traffic estimation via sparsity and low rank. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4529–4533, May 2013.

[23] N. Mastronardi, D. Kressner, V. Sima, P. Van Dooren, and S. Van Huffel. A fast algorithm for subspace state-space system identification via exploitation of the displacement structure. *J. Comput. Appl. Math.*, 132(1):71–81, 2001.

[24] M. Nasri, H. Nezamabadi-pour, and M. Maghfoori. A PSO-based optimum design of PID controller for a linear brushless DC motor. In *Proceedings of World Academy of Science, Engineering and Technology*, volume 26, pages 211–215, 2007.

[25] O. Rohat and D. Popescu. Web system for the remote control and execution of an IEC 61499 application. *Studies in Informatics and Control*, 23(3):297–306, 2014.

[26] J. Schoukens, M. Vaes, and R. Pintelon. Linear system identification in a nonlinear setting. *IEEE Control Syst. Mag.*, 36(3):38–69, June 2016.

[27] V. Sima. Fast numerical algorithms for Wiener systems identification. In V. Barbu, I. Lasiecka, D. Tiba, and C. Varsan, editors, *Analysis and Optimization of Differential Systems*, pages 375–386, Boston/Dordrecht/London, 2003. Kluwer Academic Publishers.

[28] V. Sima. Performance investigation of SLICOT Wiener systems identification toolbox. In P. Van den Hof, B. Wahlberg, and S. Weiland, editors, *Proceedings of The 13th IFAC Symposium on System Identification, SYSID 2003*, August 27–29, 2003, Rotterdam, The Netherlands, pages 1345–1350. Omnipress, 2003.

[29] V. Sima. Efficient data processing for subspace-based multivariable system identification. In *Proceedings of IFAC Workshop on Adaptation and Learning in Control and Signal Processing (ALCOSP 2004), and IFAC Workshop on Periodic Control Systems (PSYCO 2004)*, August 30th – September 1st, 2004, Yokohama, Japan, pages 871–876, 2004.

[30] V. Sima. Computational experience with a modified Newton solver for continuous-time algebraic Riccati equations. In J.-L. Ferrier, O. Gusikhin, K. Madani, and J. Sasiadek, editors, *Informatics in Control Automation and Robotics*, volume 325 of *Lecture Notes in Electrical Engineering*, chapter 3, pages 55–71. Springer International Publishing, 2015.

[31] V. Sima and P. Benner. Fast system identification and model reduction solvers. In B. Andrievsky and A. Fradkov, editors, *Proceedings of the 9th IFAC Workshop "Adaptation and Learning in Control and Signal Processing" (ALCOSP 2007)*, August, 29–31, 2007, Saint Petersburg, Russia. Curran Associates, Inc., 2007.

[32] V. Sima, D. M. Sima, and S. Van Huffel. High-performance numerical algorithms and software for subspace-based linear multivariable system identification. *J. Comput. Appl. Math.*, 170(2):371–397, 2004.

[33] V. Sima and S. Van Huffel. Performance investigation of SLICOT system identification toolbox. In *Proceedings of the European Control Conference, ECC 2001*, 4–7 September, 2001, Seminário de Vilar, Porto, Portugal, pages 3586–3591, 2001.

[34] A. Skowroński and J. Werewka. A quality attributes approach to defining reactive systems solution applied to cloud of sensors. In *2015 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 789–795, Sept 2015.

[35] M. Sorouri, S. Patil, and V. Vyatkin. Distributed control patterns for intelligent mechatronic systems. In *Proceedings of the 2012 10th IEEE International Conference on Industrial Informatics (INDIN)*, 25–27 July 2012, Beijing, China, pages 259–264. IEEE, 2012.

[36] P. Tabaghi and M. Azimi-Sadjadi. Class-preserving manifold learning for detection and classification. In *2015 International Joint Conference on Neural Networks (IJCNN)*, 12–17 July, pages 1–6. IEEE, 2015.

[37] J. Wang, Y. Zhou, K. Duan, J. J.-Y. Wang, and H. Bensmai. Supervised cross-modal factor analysis for multiple modal data classification. In *IEEE International Conference on Systems, Man, and Cybernetics*, 9–12 Oct., pages 1882–1888, Hong Kong, 2015.

[38] C. h. Yang and V. Vyatkin. Design and validation of distributed control with decentralized intelligence in process industries: A survey. In *2008 6th IEEE International Conference on Industrial Informatics*, pages 1395–1400, July 2008.

[39] C. h. Yang and V. Vyatkin. Transformation of Simulink models to IEC 61499 function blocks for verification of distributed control systems. *Control Engineering Practice*, 20(12):1259–1269, 2012.

[40] L. H. Yoong, P. S. Roop, Z. E. Bhatti, and M. M. Y. Kuo. IEC 61499 in a nutshell. In *Model-Driven Design Using IEC 61499*, page 1733. Springer International Publishing, 2015.