Program PARTENERIATE - Proiecte colaborative de cercetare aplicativă - Competiţie 2013

RST - Raport ştiinţific şi tehnic

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.

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 [3]: 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. Lucrarea [3] tratează detaliat FDD Bayesiană, care oferă un cadru teoretic foarte general, extensibil la numeroase clase importante de probleme de estimare distribuită, care depăşesc zona estimării liniar-Gaussiene şi filtrării Kalman. Se 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.

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, [16]). 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 plus, aceste tehnici necesită mai multe date decât în cazul liniar şi un efort de calcul mult mai mare. În [14] 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. Se propune o procedură care furnizează informaţie suplimentară pentru a garanta utilitatea investirii adiţionale de timp şi resurse financiare şi umane într-o abordare neliniară.

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, de pildă, în [16]. 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 [15] (recomandaţi în contextul recuperării după o funcţionare anormală).

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 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*, definită prin,

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

unde e_1, e_2, …, e_n sunt citiri ale senzorilor, iar $\bar{e} = \{e_1, e_2, \ldots, e_n\}$. 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

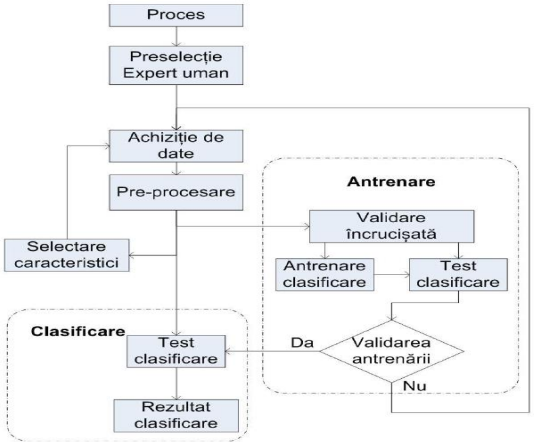

Figura 1: Algoritmul de clasificare.

astfel încât în situaţia curentă să se poată beneficia de experienţa căpătată şi însuşită din situaţiile anterioare. Schema unui astfel de algoritm de captură şi fuziune de cunoştinţe este prezentată în Fig. 1.

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. Selectarea caracteristicilor este diferită de extragerea caracteristicilor şi se efectuează prin metode specifice precum "Principal component analysis" [6], "Singular value decomposition" [9], "Manifold learning" [18] şi "Factor analysis" [19], 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.

Pentru antrenarea corectă a clasificatorului este utilizată validarea încrucişată. 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. Eşantioanele prea apropiate în ambele seturi, de antrenare şi de testare, trebuie eliminate din procedura de validare. Odată ce este obţinut un clasificator corespunzător, acesta poate fi implementat şi folosit ca atare în faza de clasificare efectivă. Precizia clasificatorului poate fi îmbunătăţită introducând 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.

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 la date 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 pentru a prezice date noi în etapa de antrenare.

Funcţii nucleu. O abordare populară pentru învăţarea bazată pe similaritate este tratarea perechilor de similarităţi ca produse scalare într-un spaţiu Hilbert. 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*

Fiind dat un nucleu care corespunde unui vector de trăsături ϕ, normalizarea nucleului $\hat{k}(x, y)$ corespunde vectorului de trăsături dat de $x \mapsto \phi(x) \mapsto \phi(x)/\|\phi(x)\|$. Normalizarea datelor ajută la îmbunătăţirea performanţelor învăţării automate pentru diferite aplicaţii practice. 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*

Prin combinarea mai multe funcţii nucleu, trăsăturile sunt scufundate într-un spaţiu dimensional mai mare, îmbunătăţindu-se performanţa clasificatorului prin creşterea spaţiului de căutare. 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. O altă posibilitate de a obţine o combinare este de a înmulţi funcţiile nucleu.

Clasificatori liniari

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

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 definiţie:

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

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*.

b. *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 are simultan 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 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, implicând 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, covarianţa vectorilor iniţiali este uneori 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. 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. 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. Nefiind o metodă de clasificare parametrică, nu există parametri care trebuie să fie învăţaţi. Ca atare, 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, cel mai adesea distanţa euclidiană. 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 [11]. Algoritmul PSO este o metodă rapidă, robustă şi uşor de implementat, capabilă să găsească optimul global în probleme de optimizare continue neliniare [11, 8, 1, 12, 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, 8, 13].

Fie o problemă de minimizare pentru o funcţie f, $f(X) = f(x_1, x_2, \ldots, x_D)$, unde $f : \mathbb{R}^D \to \mathbb{R}$ şi D este numărul total de variabile. Trebuie să găsim X^* pentru care $f(X^*) \leq f(X),\ \forall X \in S$, 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 $i = 1, 2, \ldots, P$, $t = 1, 2, \ldots, N$,

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

unde: 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, iar $Gbest^t$ 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]):

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

$$x_{ij}^{t+1} = x_{ij}^t + v_{ij}^{t+1}, \quad i = 1, 2, \ldots, P,\ j = 1, 2, \ldots, D,\ t = 1, 2, \ldots, N, \tag{3}$$

unde: v_{ij}^t şi x_{ij}^t sunt vectori ce stochează viteza, respectiv, 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, iar 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 $X^* = Gbest = (Gbest_1, Gbest_2, \ldots, Gbest_D)$. 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 [11] 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 [12]. 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].

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. Utilizarea unor diferite baze de date cuplate pentru monitorizarea evenimentelor de alertare a favorizat dezvoltarea şi implementarea unor sisteme integrate pentru monitorizarea informaţiilor şi 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 [7].

În Fig. 2 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 altă buclă 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);

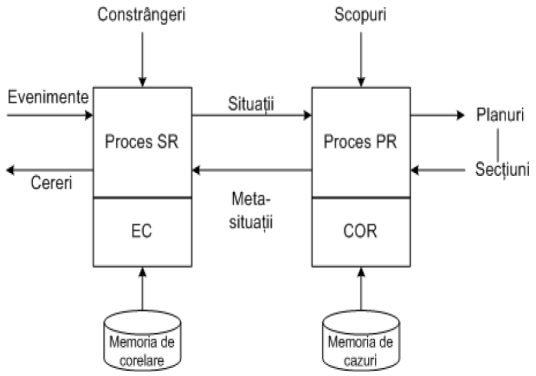

Figura 2: Relaţii reactive în procesul de luare a unei decizii.

- 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 ISDM folosind un DSS, care acţionează prin efortul cooperativ al mai multor agenţi inteligenţi în cadrul unei structuri multiagent, poate realiza:

- Monitorizare: observarea mediului şi detectarea comportamentelor problematice;
- Generare de alarme: activarea unei alarme dacă apare o situaţie critică;
- Avertizare, 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: Analizând informaţia, clasificatorul alege starea sistemului monitorizat;
- Diagnoză: Explicaţie în termeni cauzali a evenimentelor şi situaţiilor inacceptabile;
- Planificarea acţiunii: Stabilirea unei posibile succesiuni 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. Sarcinile locale vor fi de o complexitate mai redusă, dar interdependente. Sarcina de coordonare într-un astfel de sistem se referă la managementul dependenţelor între sarcinile locale, realizat de regulă printr-o metodologie orientată pe cunoştinţe. În procesul de rezolvare de probleme asociat unei sarcini, fiecare pas poate stabili mai multe sub-sarcini, care la rândul lor trebuie rezolvate recursiv de metode mai simple, până când se ajunge la o sarcină elementară, direct îndeplinită.

3.2 Metode de analiză decizională multicriterială utilizate în detecţia hazardurilor

3.2.1 Consideraţii generale

Formularea problemei. Ipoteza iniţială a unei probleme de decizie multicriterială constă în existenţa a m criterii şi n variante notate $C_1, C_2, \ldots, C_m$, respectiv $A_1, A_2, \ldots, A_n$ (ambele finite). Particularitatea acestor metode este dată de existenţa matricei decizionale, care este prezentată sub forma unei tabele (vezi Tab. 1). 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

Tabela 1: Forma generală a unei tabele decizionale

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

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:

- 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.
- Existenţa criteriilor conflictuale. (Criteriile multiple se află uneori în situaţii conflictuale.)
- 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 metode sunt enumerate mai jos [4]. *Analiza bazată pe argumentele pro şi contra* este o metodă de comparare calitativă în care sunt identificate lucrurile bune (pro) şi lucrurile rele (contra) 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.

3.2.2 Metoda Analitică Ierarhizată

Metoda de analiză multicriterială poate să crească în complexitate prin rafinarea criteriilor, introducerea de subcriterii şi optimizarea procesului de alocare a ponderilor.

Metoda Analitică Ierarhizată (MAI) este o abordare de luare a deciziilor multicriteriale care a atras interesul multor cercetători. 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 SC= $\{\text{SC}_1, \text{SC}_2, \ldots, \text{SC}_n\}$, 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 1, debutul algoritmului MAI este de constituire a matricei decizionale comparative A, cu elementele a_{ij}. Elementele diagonalei principale, a_{ii}, pot fi interpretate ca rezultat al cuantificării între două variante

identice (V_1 şi V_1, V_2 şi V_2, ..., V_n şi V_n). Aşadar, se poate considera că $a_{ii} = 1$. Reciprocitatea cuantificării poate fi enunţată de asemenea astfel: fiind dată cuantificarea a_{ij} a două variante V_i şi V_j, $i \neq j$, reciproca acesteia va fi exprimată de relaţia: $a_{ij} = 1/a_{ji}$.

Considerând cunoscute datele primare de intrare definite de mulţimile C, SC, precum şi cuantificările a_{ij} asociate acestora, pasul următor al algoritmului constă în normalizarea matricei decizionale A, pentru ca punctajele diferitelor criterii să poată fi comparate. MAI nu menţine ordinea de mărime relativă, ci stabileşte punctajele clare ale opţiunilor în intervalul [0,1]. Pentru aceasta, se aplică normalizarea brută,

$$a_{ij}^* = a_{ij} / \sum_{k=1}^{n} \sum_{l=1}^{n} a_{kl}. \tag{4}$$

Al doilea pas în algoritmul MAI calculează ponderile parţiale, exprimate, de regulă, sub formă de procente. Se foloseşte formula,

$$w_n = \frac{1}{n} \sum_{k=1}^{n} a_{kn}^*, \tag{5}$$

unde w_n reprezintă gradul de preferinţă (ponderea alternativei în preferinţele decidenţilor) şi n rangul matricei decizionale. Ultimul pas al MAI este realizarea matricei de putere care va furniza clasamentul final al variantelor, reprezentat de vectorul W, al preferinţelor finale, care este vectorul propriu corespunzător valorii proprii maxime, $\lambda_{\max}$, a matricei A. Aşadar, W satisface ecuaţia $AW = \lambda_{\max} W$, cu $W \neq 0$.

Metoda puterilor permite calculul vectorului propriu W 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 sunt notaţi cu k, $k = 1, 2, \ldots, m$, iar numărul iteraţiilor este m. Fiecare pas obţine o aproximare îmbunătăţită 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 $W^1 = \left[n^{-1}, \ldots, n^{-1}, \ldots, n^{-1}\right]^T$. Pentru $k \geq 2$, se introduce vectorul $d^k = AW^{k-1}$. Aproximările succesive ale vectorului W^k sunt $W^k = d^k / \|d^k\|$. Sfârşitul calculelor este realizat când aproximările devin mai mici ca ε: $\|W^k - W^{k-1}\| < \varepsilon$.

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, pentru a oferi o interfaţă uniformă cu utilizatorul, mentenanţă uşoară 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 selectabile prin 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 [17] 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, numite 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 (date ce trebuie prelucrate şi care influenţează procesul de decizie) şi

comenzi (care trebuie executate). 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 [17] 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, iar nu pentru condiţionarea tranziţiilor din diagrama ECC.
- *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 comandată de 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 eventuale modificări dorite. 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 3. 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 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.

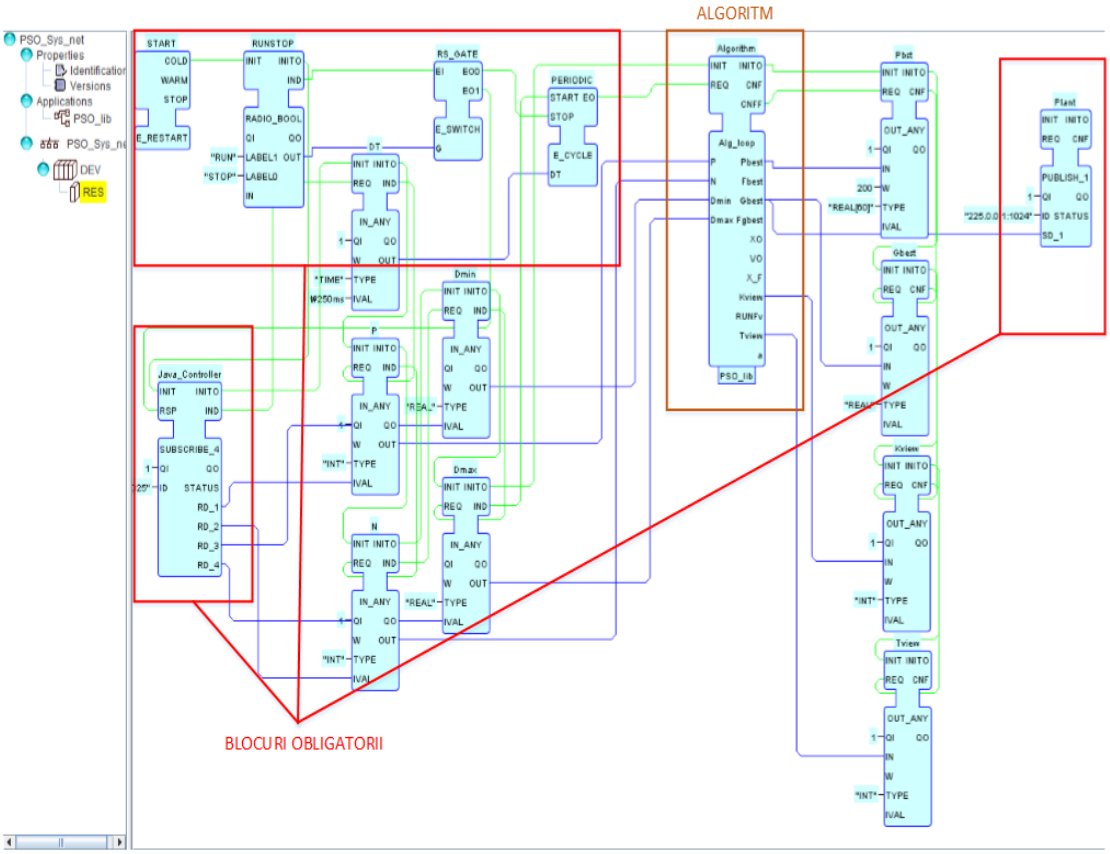

Figura 3: Exemplu de construire a unei configuraţii de sistem.

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, fiecare algoritm va fi însoţit de acea scurtă descriere. Algoritmii care implementează funcţii mai complexe (de optimizare etc.) vor putea fi executaţi în cloud. Pentru trimiterea rezultatului către un echipament aflat la distanţă, 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:

- Adresa de IP a sursei datelor şi adresa de IP a destinaţiei datelor;
- Numele serverului de OPC din/către care se face accesul/scrierea datelor;
- Calea şi numele parametrilor corespunzători datelor de intrare/ieşire necesare/rezultate.

(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.

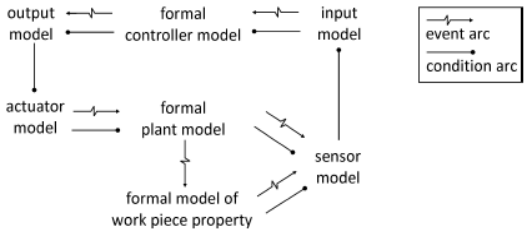

Figura 4: Schema modelării în buclă închisă a unei aplicaţii de control [10].

(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 [20]. 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.

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 [10] se abordează problema verificării şi validării aplicaţiilor de control pornind de la testarea modelului procesului.

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. 4. 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 [21]. Metoda funcţionează pe principiul mapării directe a funcţiilor bloc din Simulink pe cele specifice standardului 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. 5. Pentru a determina precis comportarea

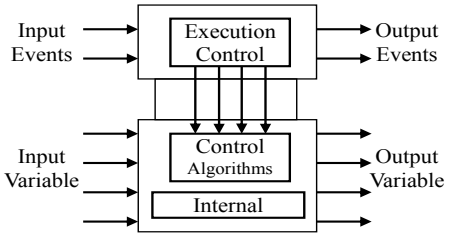

Figura 5: Reprezentarea unei funcţii bloc conform standardului IEC 61499.

unui dispozitiv interconectat într-o aplicaţie distribuită, este important să fie cunoscute regulile execuţiei unei funcţii bloc, adică, semantica [5]. 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 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 şi CLIENT/SERVER, pentru comunicaţia unidirecţională, respectiv, 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.

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 a relaţiilor dintre ele). Transmiterea de mesaje între servicii este reprezentată de conexiunile dintre funcţiile bloc.

Variabilele de date asociate unui eveniment de conexiune, în accepţiunea SOA, sunt utilizate ca parametri de intrare ai mesajului. Este normal ca operaţiile realizate prin funcţii bloc să poată fi considerate ca servicii cloud [5]. 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. 6 sunt ilustraţi, ca exemplificare, paşii necesari pentru implementarea algoritmului PSO, luând în considerare mecanismul de execuţie a funcţiilor bloc IEC 61499. În Fig. 6, 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 şi D sunt numărul total de particule, respectiv, de variabile. 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

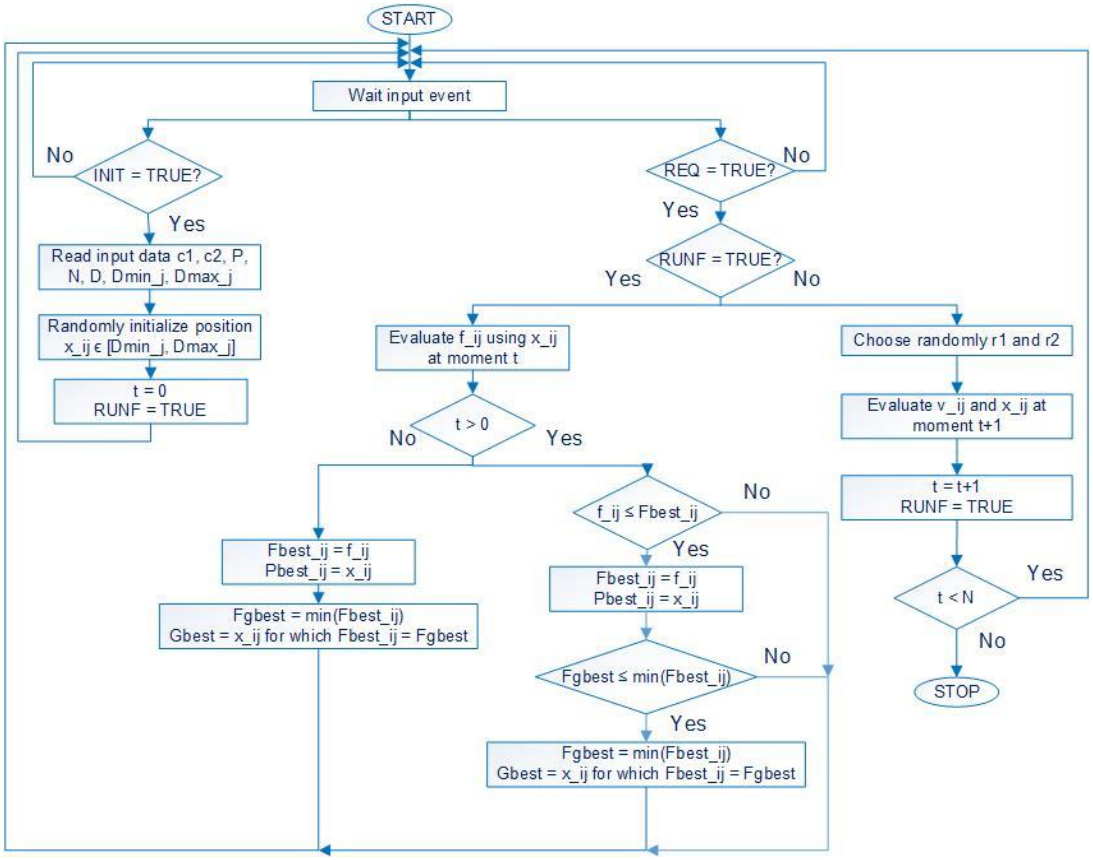

Figura 6: Diagrama algoritmului PSO.

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. Fbest şi Pbest reprezintă vectori ce reţin valorile minime ale funcţiei obiectiv la momentul t, respectiv, poziţiile corespunzătoare. 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. 7. 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 asupra spaţiului de stocare, dar nu asupra performanţei de execuţie. 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. 8. 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. 6. La recepţionarea unui eveniment REQ, algoritmul verifică valoarea variabilei RUNF. Dacă aceasta este falsă, atunci se execută funcţia

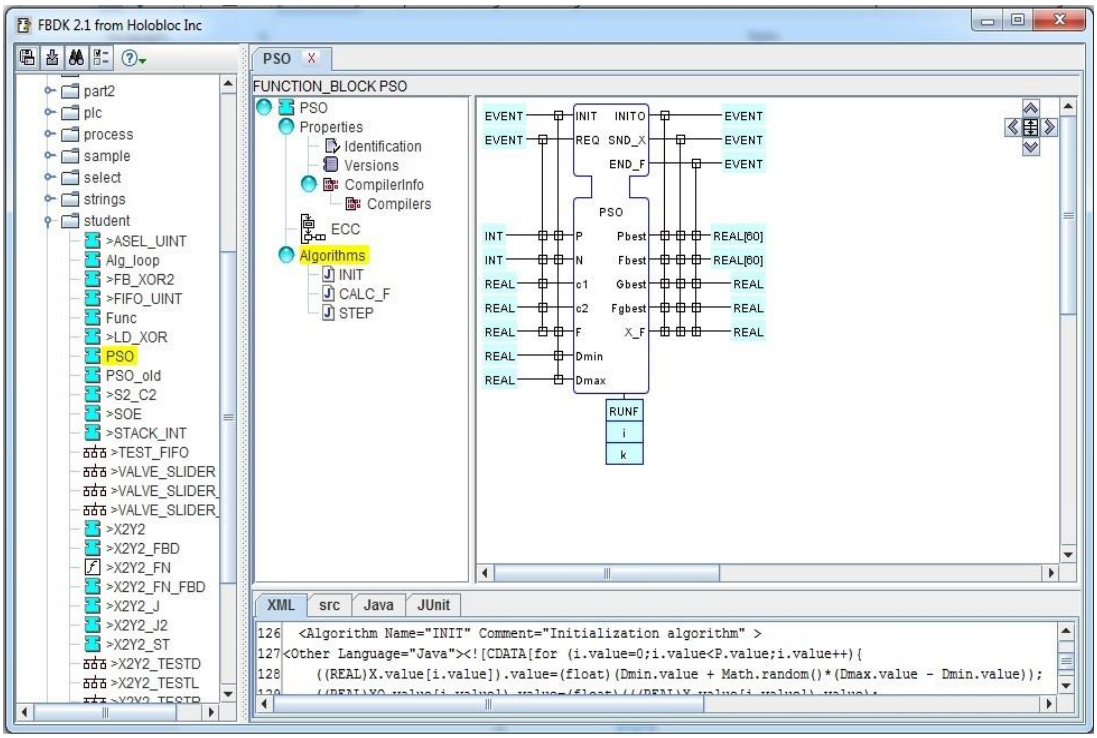

Figura 7: Structura algoritmului.

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. 6. Dacă RUNF are valoarea TRUE înseamnă că trebuie să se execute funcţia CALC_F. Aceasta corespunde secţiunii rămase din diagrama din Fig. 6. Pentru a obţine un algoritm reutilizabil şi independent de funcţia minimizată, 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. Doi dintre algoritmii 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.

```
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;
```

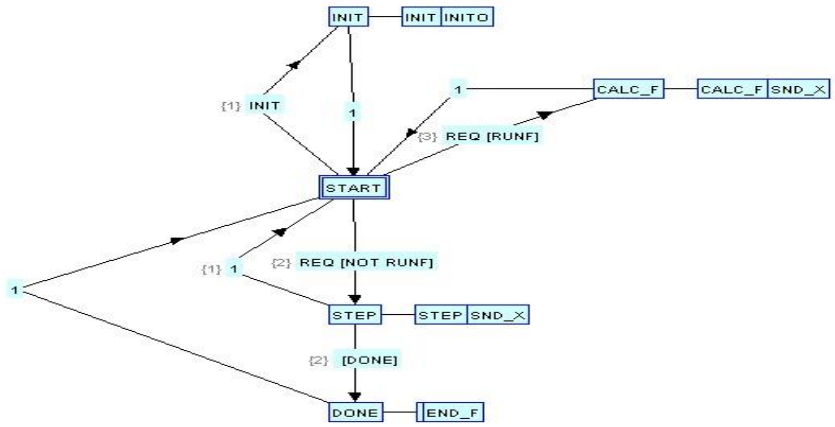

Figura 8: Diagrama de execuţie ECC pentru algoritmul PSO.

Algoritmul CALC_F are ca scop calcularea lui F pentru fiecare element al vectorului de poziţie f la fiecare iteraţie şi căutarea optimului. 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 calcula valoarea funcţiei. La prima iteraţie (t = 0) algoritmul reţine valorile lui F şi 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).

Algoritmul STEP calculează vectorii de poziţie X şi de viteză V de la iteraţia curentă utilizând (3). Pentru aceasta sunt generaţi întâi parametrii aleatori r1 şi r2. Variabila ao este folosită ca precalcul prentru simplificarea formulei pentru vectorul V. 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. Au fost adăugate câteva variabile de ieşire pentru a evalua mai bine execuţia algoritmului. Pentru execuţie, s-au folosit următorii parametri de intrare: $P = 20$, $N = 30$, $c_1 = c_2 = 2$, Dmin = 0, Dmax = 15. Funcţia obiectiv de test, Test_func, implementează formula out $= f(x) = x^2 + x + 1$. S-a analizat ieşirea algoritmului PSO, implementat în IEC 61499, pornind cu o populaţie iniţială aleatoare la momentul $t = 0$. Fig. 9 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

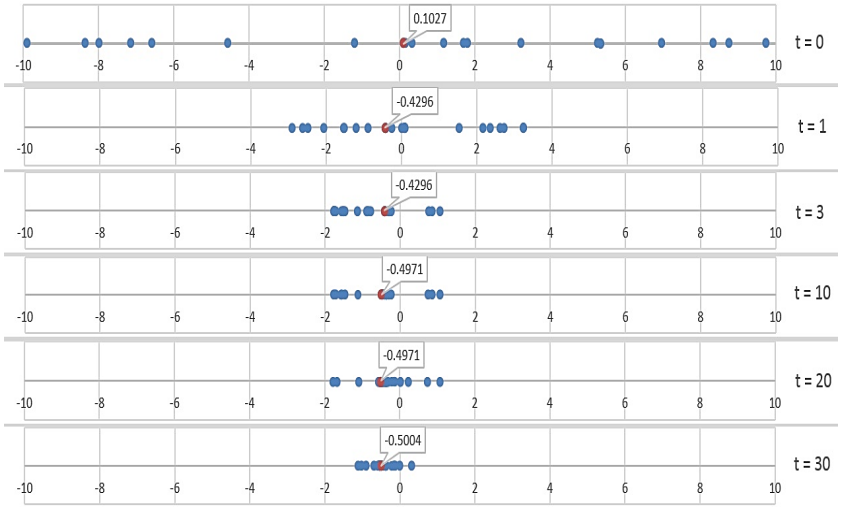

Figura 9: Mişcarea particulelor către valoarea optimă globală.

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. 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, pentru a putea compara performanţele şi rezultatele obţinute. Soluţia algoritmului implementat în MATLAB, obţinută după 14 iteraţii, a fost Gbest = -0.4998, Fgbest = 0.7500. Diferenţa între rezultate se datorează populaţiei iniţiale diferite, precum şi parametrilor aleatori r_1 şi r_2.

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. Al doilea argument, Executors.newSingleThreadScheduledExecutor(), creează un fir de execuţie independent de program, 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 care 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 indică id-ul locaţiei datelor şi 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) {  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.

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, dar 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.

În [22] 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. Ele sunt similare funcţiilor bloc de bază în sensul că sunt încapsulate prin intermediul unor interfeţe de funcţii bloc, dar 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;
- Servicii de comunicare (fie unidirecţională, implementată cu ajutorul funcţiilor bloc de tipul publish/subscribe, fie 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. 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. 10. 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;
- 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 sub-componentelor 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 planificarea şi

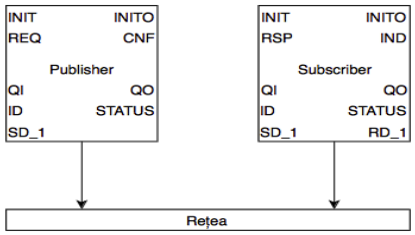

Figura 10: Funcţii bloc de interfaţă de tipul publish-subscribe.

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.
- Interfaţa de tip OPC este necesară pentru comunicarea cu procesul controlat, cum ar fi instrumente, senzori sau elemente de execuţie. Pentru comunicarea cu un sistem IEC 61499 este necesară o funcţie bloc de interfaţă (SIFB) care să implementeze standardul OPC. Aceasta ar putea fi un server OPC, dar care trebuie să fie executat de către mediul de execuţie IEC 61499. Similar, în cazul interfeţei care

implementează servicii web de tip REST, sau pentru interfaţa om-maşină, 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 sau publish-subscribe.
- 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.

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. 11). 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;
- Analiza unui fişier de tip istoric şi salvarea datelor într-o anumită tabelă din baza de date;
- Controlul execuţiei funcţiilor: pornire/oprire, creare instanţe conform cu configurarea din pagina web;
- Trimiterea unei confirmări către aplicaţia web în momentul pornirii sau opririi execuţiei 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.
- 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 are rolul de gestionare a accesului utilizatorului la resursele bibliotecii (Fig. 12). Ea va permite:

- Configurarea detaliilor de conectare la instalaţie.
- Posibilitatea de încărcare a unei serii de timp (fişier text sau xls), în scopul analizei offline.

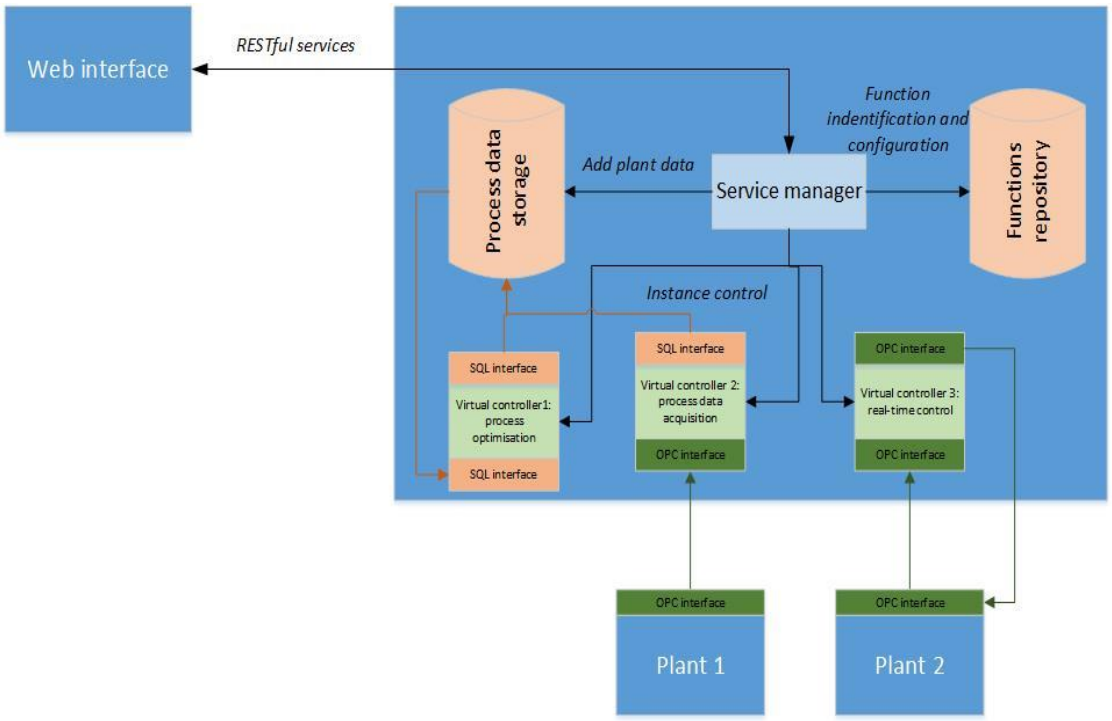

Figura 11: Gestionarea execuţiei funcţiilor (săgeţile negre reprezintă servicii RESTful, cele portocalii - funcţii de interfaţare SQL, iar cele verzi - date OPC).

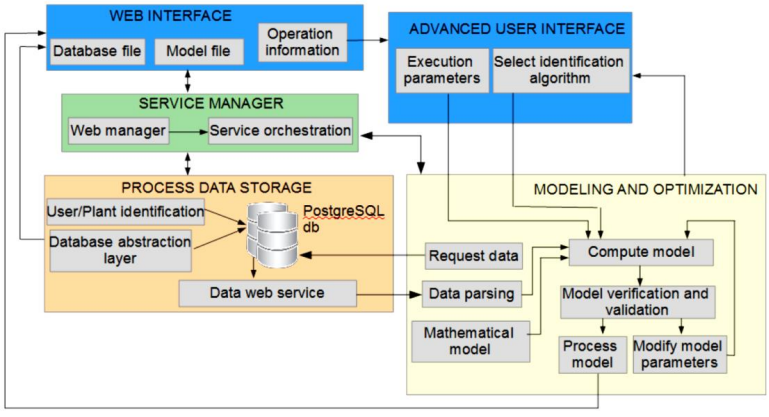

Figura 12: Gestionarea aplicaţiei web şi interconectarea cu baza de date de proces.

- 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.

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, 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.

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. 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.

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] 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.

[4] 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.

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

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

[12] 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.

[13] 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.

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

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.