Universitatea „DUNĂREA DE JOS” Galaţi
Facultatea de Automatică, Calculatoare,
Inginerie Electrică şi Electronică

TEZĂ DE DOCTORAT

Simularea în timp real a comportamentului echipamentelor pentru exploatarea sistemelor critice

- Rezumat -

ing Lucian – Florentin BĂRBULESCU

Conducător ştiinţific: Prof. dr. ing. Viorel-Nicolae MÎNZU

- Gataţi, 2013-

7028/26.03.2013

Către

_____

_____

Universitatea "Dunărea de Jos " din Galaţi vă face cunoscut că în data de**19.04.2013**, ora**11.00, în sala Y405 a Facultăţii de Automatică, Calculatoare, Inginerie Electrică şi Electronică**, va avea loc susţinerea publică a tezei de doctorat intitulată: **"SIMULAREA ÎN TIMP REAL A COMPORTAMENTULUI ECHIPAMENTELOR PENTRU EXPLOATAREA SISTEMELOR CRITICE"**, elaborată de domnul/doamna **BĂRBULESCU LUCIAN-FLORENTIN**, în vederea conferirii titlului ştiinţific de doctor în domeniul de doctorat **Ingineria sistemelor.**

Comisia de doctorat are următoarea componenţă :

1. Preşedinte	**Conf.univ.dr.ing. Emilia PECHEANU** Universitatea "Dunărea de Jos" din Galaţi
2. Conducător de doctorat	**Prof.univ.dr.ing. Viorel-Nicolae MÎNZU** Universitatea "Dunărea de Jos" din Galaţi
3. Referent oficial	**Prof.univ.dr.ing. Simona CARAMIHAI** Universitatea POLITEHNICA Bucureşti
4. Referent oficial	**Prof.univ.dr.ing. Marin LUNGU** Universitatea din Craiova
5. Referent oficial	**Prof.univ.dr.ing. Sergiu CARAMAN** Universitatea "Dunărea de Jos" din Galaţi

Cu această ocazie vă transmitem rezumatul tezei de doctorat, şi vă invităm să participaţi la susţinerea publică. În cazul în care doriţi să faceţi eventuale aprecieri sau observaţii asupra conţinutului lucrării, vă rugăm să le transmiteţi în scris pe adresa universităţii, str. Domnească nr. 47, 800008 Galaţi, Fax 0236 / 461353.

Rector,
Prof. univ. dr. ing. Iulian Gabriel BÎRSAN

RECTORAT

Cuprins

1. Introducere

Automatizarea proceselor este, în acest moment, prezentă în toate domeniile unei economii performante. Nu se mai poate discuta de performanţă fără să se ia în calcul folosirea unor sisteme automate capabile să efectueze rapid şi eficient diferite operaţii mai simple sau mai complicate şi, cel mai important, fără erori. Dezvoltarea unui sistem automat este însa foarte costisitoare şi nu se poate garanta în totalitate că produsul final oferă avantaje suficient de mari cât să acopere şi să justifice costurile necesare pentru realizarea sa. În plus, multe companii evită să investească sume mari în dezvoltarea unor sisteme automate daca nu au garanţia unor avantaje, cel puţin pe termen lung. În general, înainte de a achiziţiona un sistem automat, sau finanţa dezvoltarea acestuia, beneficiarii doresc să vadă un prototip sau o simulare a funcţionării pentru a putea să îşi facă cel puţin o idee asupra produsului final. Astfel, realizarea de simulatoare pentru sistemele automate devine un pas important în dezvoltarea acestora.

Simulatoarele au însă mult mai multe utilizări: ele pot fi folosite, pentru oferirea unei imaginii de ansamblu asupra funcţionării unei instalaţii, pentru testarea funcţionării sau rezistenţei unor componente sau echipamente, pentru instruirea personalului care foloseşte anumite instalaţii industriale etc. Astfel, realizarea de simulatoare este un pas important în dezvoltarea unor instalaţii industriale şi este, în general, o cerinţă impusă şi de către beneficiari, dar are şi anumite dezavantaje legate în general de costul de producţie.

De multe ori realizarea unui simulator este mult mai costisitoare decât realizarea instalaţiei propriu-zise şi astfel costul de producţie poate ajunge la o valoare semnificativa şi poate deveni un impediment. O caracteristică importantă a unui simulator este capacitatea acestuia de a imita perfect comportamentul unei instalaţii. Pe măsură ce complexitatea instalaţiilor a crescut, a crescut şi necesitatea ca un simulator sa poată executa anumite operaţii într-un timp mai scurt. În cazul simulatoarelor software durata de execuţie se poate micşora fie prin îmbunătăţirea suportului hardware pe care rulează (procesor mai rapid şi cu mai multe nuclee, memorie mai rapidă şi mai mare etc.), fie prin folosirea resurselor mai multor maşini (simulatoare distribuite). Prima varianta are avantajul unei structuri mai simple a simulatorului, dar poate impune costuri mai mari (deoarece componentele rapide – procesor, memorie, hard-disk etc. – sunt foarte scumpe), timp de execuţie crescut şi este limitată datorită nivelului tehnologiei [Egea-Lopez, 2009]. Cea de-a doua variantă presupune legarea mai multor maşini fizice mai slabe prin intermediul unui mediu de comunicaţie comun (de ex. o reţea de calculatoare), având astfel avantajul măriri puterii de calcul la un cost relativ mic, dar are dezavantajul complexităţii nivelului software care trebuie sa fie proiectat şi implementat în aşa fel încât să poată fi rulat într-un sistem distribuit.

Ideal ar fi realizarea unui simulator ce poarte rula atât în cadrul unui sistem distribuit, dar şi pe un sistem centralizat, folosindu-se astfel, după caz, de avantajele ambelor variante. Un astfel de sistem este propus în prezenta lucrare.

Principiul de funcţionare al simulatorului se bazează pe folosirea unor componente simple, numite simulatoare elementare, ce se interconectează între ele prin intermediul unor elemente de legătură, numite canale de comunicaţie, rezultând astfel o aplicaţie ce poate rula pe una sau mai multe maşini. Aceste canale de comunicaţie vor putea fi implementate folosind diferite tehnologii (memorie partajata, cozi de mesaje, TCP/IP, JMS etc.) fapt ce introduce o noutate fata de alte abordări asemănătoare ce sunt limitate la folosirea unor tehnologii specifice Java, cum ar fi, de exemplu, RMI [Ferscha&Richter, 1997].

Toate componentele sistemului vor fi implementate în Java datorita avantajelor oferite de acest limbaj şi anume orientarea pe obiecte, folosirea unor tipuri de date stricte, verificarea limitelor

tablourilor, avantaje ce îl apropie de limbajul ADA ce este folosit în mod tradiţional în sistemele critice[Brogsol, 2009]. Dezavantajele folosiri Java în cazul sistemelor de timp real, şi anume impredictibilitatea execuţiei, pot fi eliminate prin folosirea unor implementări speciale de maşini virtuale ce rulează într-un sistem de operare Real-Time Linux [Auerbach el all., 2007] sau prin folosirea unor arhitecturi hardware speciale, cum ar fi de exemplu Java Optimized Processor (JOP) [Schoeberl, 2008], un procesor ce rulează nativ Bytecode.

În capitolul 2 se prezintă câteva noţiuni teoretice referitoare la simularea sistemelor dinamice punându-se accentul pe avantajele oferite dar, în acelaşi timp, evidenţiind şi dezavantajele ce rezultă şi nu pot fi ignorate. În continuare se oferă câteva informaţii despre simularea sistemelor dinamice cu evenimente discrete, prezentându-se pe scurt cele doua abordări principale şi anume simularea condusă de timp şi simularea condusă de evenimente. În finalul capitolului sunt enumerate şi detaliate abordările paralele/distribuite folosite pentru simularea condusa de evenimente a sistemelor dinamice cu evenimente discrete.

În capitolul 3 se propune structura generala a unui sistem de simulare distribuit ce foloseşte doua categorii de module software numite Simulatoarele Elementare (SE) şi, respectiv, Canale de Comunicaţie (CC) şi se prezintă funcţiile generale pe care acestea le îndeplinesc. Se prezintă şi un algoritm ce este folosit pentru stabilirea numărului optim de elemente de tip CC astfel încât să se eficientizeze cantitatea de memorie utilizată şi timpul de execuţie al simulării, dar şi proiectarea generala a unui modul software ce poate fi folosit pentru interfaţa cu operatorii umani şi a unui modul software ce poate fi folosit pentru comunicaţia cu echipamente fizice. De asemenea, se proiectează şi se implementează un modul software folosit pentru citirea şi interpretarea fişierului de configuraţie al simulatorului, iar, în finalul capitolului, se proiectează şi se implementează un simulator, ce foloseşte structura prezentata anterior, pentru o instalaţie industriala simpla, formata din doua bazine ce conţin cantităţi diferite de apa şi care sunt interconectate între ele prin intermediul unui robinet.

În capitolul 4 se realizează o analiză a simulatoarelor elementare (SE), văzute ca sisteme dinamice, pentru identificarea funcţionalităţilor comune şi se proiectează şi se implementează un algoritm de serializare simplu şi eficient ce este folosit pentru transportul valorilor variabilelor de intrare şi de ieşire între SE. Se proiectează şi se implementează un modul software, numit Simulatorul Elementar Abstract (SEA), ce însumează funcţionalităţile comune ale SE şi se realizează o analiză privind eficienţa şi viteza de execuţie a SEA. Pornind de la implementarea SEA se proiectează şi se implementează două module software, unul folosit pentru interacţiunea cu operatorii umani, şi unul folosit pentru generarea unor mesaje la intervale de timp regulate.

În capitolul 5 sunt definite regulile generale de funcţionare ale canalelor de comunicaţie (CC), sunt prezentate interfeţele ce trebuie folosite de modulele software ce oferă implementări şi sunt oferite definiţii pentru canalele de comunicaţie locale (CCL) şi canalele de comunicaţie distribuite (CCD). Sunt detaliate doua implementări pentru CCL, una bazată pe memorie partajată şi una bazată pe cozi de mesaje şi se realizează o analiză privind eficienta şi viteza de execuţie a acestora, precum şi doua implementări pentru CCD, una bazată pe un protocol simplu peste TCP/IP şi una bazată pe JMS şi se realizează o analiză privind eficienta şi viteza de execuţie a acestora.

În capitolul 6 se realizează o analiză a funcţionării unei ecluze navigabile şi se stabilesc subsistemele pentru care se realizează Simulatoare Elementare. Se proiectează şi se implementează module software, bazate pe SEA, pentru simularea comportamentului nivelului amonte sau aval, a porţii plane, a porţii buscate, a vanelor de umplere sau de golire şi a sasului unei ecluze navigabile. Pentru fiecare modul software se realizează o componenta grafică ce este folosită pentru realizarea

interfeței cu utilizatorul. În plus, se proiectează și se implementează un modul software pentru controlul și comanda subsistemelor enunțate mai sus.

În capitolul 7 sunt prezentate concluziile, contribuțiile personale precum și direcțiile viitoare de dezvoltare ale prezentului sistem de simulare.

2. Metode și tehnici de simulare

2.1. Scopul, avantajele și dezavantajele simulării

În lucrările de specialitate [Chung, 2004] sunt prezentate scopurile, avantajele, dezavantajele dar și alte considerente ale simulării. Astfel, analiza și simularea diferitelor tipuri de sisteme este realizata pentru atingerea următoarelor scopuri:

- Obținerea de informații referitoare la funcționarea unui sistem.
- Dezvoltarea de politici de exploatare pentru îmbunătățirea performanțelor sistemului.
- Testarea unor concepte și/sau sisteme înainte de implementare.
- Obținerea de informații fără a perturba funcționarea sistemului.

Pe lângă faptul ca poate fi folosita pentru a răspunde la cerințele enunțate anterior, simularea oferă și alte beneficii, cum ar fi:

- Experimentarea în timp redus.
- Oferirea de modele ușor de demonstrat.
- Se păstrează controlul asupra experimentului.

Deși simularea oferă foarte multe avantaje, exista și câteva dezavantaje de care trebuie ținut cont. Acestea nu sunt direct legate de modelarea și analiza unui sistem, ci de rezultatele așteptate în urma simulării. Aceste dezavantaje includ următoarele:

- Simularea nu poate oferi rezultate precise atunci când datele de intrare sunt imprecise.
- Simularea nu poate oferii răspunsuri simple la probleme complicate.
- Simularea nu poate rezolva problemele singura.

Pe lângă avantajele și dezavantajele enunțate anterior trebuie sa se tina cont și de alte considerente atunci când se începe realizarea unui model de simulare. Acestea pot influenta decizia daca este oportun sau nu realizarea proiectului. Amintim următoarele:

- Construirea modelului de simulare poate necesita pregătire speciala.
- Modelarea, simularea și analiza pot fi costisitoare.
- Rezultatele simulării conțin multe date statistice.

2.2. Sisteme Dinamice

Cuvântul "sistem" a devenit foarte popular în ultimii ani, fiind folosit, nu numai în inginerie, dar și în știință, economie, sociologie, și chiar și în politică. Un sistem este definit ca o combinație de componente care acționează împreună pentru a realiza un anumit obiectiv. Un pic mai filozofic, un sistem poate fi înțeleasă ca o parte conceptual izolata a universului și care are un interes pentru noi. Alte părți ale universului, care interacționează cu sistemul formează sistemul vecin

Obiectivul principal al analizei sistemelor este acela de a anticipa modul în care sistemul va răspunde la diferite valori ale intrărilor și cum se vor modifica aceste răspunsuri în funcție de variații ale parametrilor săi. Pentru a realiza acest lucru inginerii erau nevoiți sa realizeze sisteme prototip pentru a le testa. În timp ce datele obținute din sistemele fizice prototip sunt foarte bune, costurile în timp și bani necesare pentru a le obține pot fi prohibitive. Mai mult, modelele matematice sunt mult mai flexibile decât prototipurile fizice și permit ajustarea rapida a sistemului pentru a se obține rezultate mai bune. De aceea unul dintre primii pași în analiza sistemului este reprezentat de stabilirea

unui model matematic adecvat care sa poată fi folosit pentru obţinerea de informaţii echivalente cu acelea care ar rezulta de la mai multe prototipuri fizice. În acest fel, chiar daca un prototip fizic va fi construit pentru a se verifica modelul matematic, cei care construiesc sistemul au scutit deja foarte mult timp şi resurse.

Un model matematic este un set de ecuaţii care descriu complet relaţiile dintre variabilele sistemului. El este folosit ca un instrument în dezvoltarea proiectului sau a algoritmilor de control şi tema principala pentru care este folosit are implicaţii în alegerea unei forme particulare a modelului sistemului. Cu alte cuvinte un model poate fi considerat un instrument specializat dezvoltat special pentru a anumita aplicaţie. Construirea de modele matematice universale, chiar şi pentru sisteme cu complexitate moderata, este nepractic şi neeconomic.

Modelele matematice pot fi grupate în funcţie de diverse criterii [Kulakovsky et al., 2007], şi anume: aplicabilitatea principiului superpoziţiei, dependent de coordinate spaţiale şi temporale, variaţia parametrilor în timp şi continuitatea variabilelor independente. Pornind de la aceste aspect modelele se pot împărţii în diverse categorii [Ljung, 1994], si anume: *Liniare – Neliniare, Concentrate – Distribuite, Determinist – Stocastic, Dinamic – Static, Continue – Discrete.*

Sistemele continue (cu evoluţie continua în timp) pot fi modelate într-un spaţiu continuu al stărilor folosind aparatul ecuaţiilor diferenţiale sau cu derivate parţiale, ce urmăresc variaţia continua a parametrilor. Metodele analitice oferă un model reprezentat în general prin formule uşor de utilizat şi rezultate exacte. Uneori, pentru sisteme complexe şi/sau modele foarte detaliate, obţinerea acestor formule poate deveni foarte dificila, chiar imposibila; în astfel de cazuri, simulările ce aproximează dinamica sistemului prin oferirea de ipostaze non-continue incrementale, pot fi utile, deşi sunt mai dificil de realizat şi oferă doar rezultate aproximative. Pentru Sistemele Dinamice cu Evenimente Discrete (SDED), ca sisteme cu evoluţie discreta în timp, şi vor fi referite în special cele complexe, greu accesibile şi dificil de controlat, însăşi dinamica originala este de acest tip, incrementarea timpului se face nu cu durate fixe, ci variabile, conform cu regulile de succesiune ale evenimentelor.

2.2.1. Abordări principale în simularea secvenţială a SDED.

Din punct de vedere istoric se pot observa doua abordări principale în simularea SDED, diferite ca importanta, cel puţin din punctul de vedere al simulării secvenţiale[Mocanu, 1999]:

Simularea condusa de timp

Aceasta abordare implica existenta unui ceas de timp (orologiu) central, ce progresează (avansează) în unităţi de timp incrementale fixe pentru o aceeaşi problema tratata. La fiecare incrementare a ceasului de timp, lista arbitrar ordonata a evenimentelor este parcursa, iar fiecare eveniment al cărui moment de timp asociat (marcat) corespunde cu valoarea curentă a orologiului, este simulat (executat).

Simularea condusa de evenimente

În aceasta abordare existenta unui ceas de timp global, deşi avuta în vedere, nu prezintă aceeaşi importanta ca în cazul anterior. Parametrul timp este doar asociat apariţiei evenimentelor, iar progresia temporala a simulării este implicita, pe măsură ce se avansează în tratarea evenimentelor. Planificarea corecta a execuţiei evenimentelor impune cu necesitate includerea lor într-o lista ordonata pe baza timpilor de apariţie asociaţi - pe care o vom numi "*lista evenimentelor viitoare*", denumire corecta pentru toate evenimentele din lista, minus eventual cel aflat în fruntea listei, ce poate fi deja în curs de tratare şi care oricum este desemnat în mod particular ca "*eveniment iminent*". Toate evenimentele cu excepţia evenimentului iminent sunt de fapt evenimente potenţiale, deoarece ele pot decalate, modificate sau chiar anulate prin simularea evenimentului în curs.

2.3. **Simularea paralela/distribuita a SDED.**

Așa cum este sugerat de nume, simularea paralela/distribuita a unui SDED presupune folosirea unui set de unități de procesare interconectate între ele. Diferența între paralel și distribuit nu este foarte mare și poate fi concluzionata în câteva cuvinte astfel: simularea paralela presupune folosirea mai multor nuclee de procesare ce împart aceiași zonă de memorie în timp ce simularea distribuită presupune folosirea unei memorii distribuite, de exemplu mai multe calculatoare interconectate printr-o rețea. În prezent, platformele de simulare folosite, sunt foarte adesea o combinație a celor două, așa cum se întâmplă în clusterele de calculatoare bazate pe ratele de echipamente cu mai multe procesoare (multi-CPU) și/sau mai multe nuclee de procesare (multi-core)[D'Angelo, 2011].

Indiferent de situație, simularea paralela/distribuita presupune folosirea puterii de calcul a mai multor nuclee de procesare aflate fie pe aceiași mașină fizică, fie pe mașini diferite, fapt ce oferă o serie de avantaje[Fujimoto, 2000], cum ar fi: *Timp de execuție redus, Distribuire geografica, Integrarea simulatoarelor ce sunt executate pe mașini aparținând unor producători diferiți, Toleranta la erori.*

Din punct de vedere tehnic, principala diferență dintre simularea secvențială și simularea paralela/distribuita este lipsa unei stări globale, care este reprezentarea sistemului simulat într-un model sintetic[D'Angelo, 2011]. Lipsa unei stări globale și prezența unei rețele care interconectează diferite părți ale simulatorului are unele consecințe importante[D'Angelo, 2011]:

- Modelul care reprezintă sistemul simulat trebuie să fie împărțit în componente
- Rezultatele unei simulări paralele / distribuite sunt corecte numai în cazul în care sunt identice cu cele care s-ar fi obținut folosind a abordare secvențiala.
- Fiecare componentă a simulatorului va produce actualizări ale stării sale, care sunt, eventual, relevante pentru alte componente.

2.3.1. Simularea paralela/distribuita prin interacțiunea proceselor.

Prin cele prezentate până acum s-a avut în vedere o introducere în problematica specifica simulării paralele / distribuite bazata pe o abordare aproape identica cu cea utilizată pe scara larga în simularea secvențială: planificarea de evenimente. Comun însa tuturor strategiilor de simulare ce aplica distribuția la nivelul evenimentelor este orientarea lor către împărțirea sarcinii globale în procese logice, ce interacționează și comunica între ele. O simulare prin interacțiunea proceselor logice poate fi astfel privita ca o partajare a domeniului spațiu - timp între procesele logice.

În ultima perioada au fost propuse și studiate mai multe variante de algoritmi de sincronizare, din care amintim[D'Angelo, 2011]:

- *Pas de timp constant (time-stepped)*: Timpul simulat este împărțit în intervale de dimensiune fixa și fiecare unitate de procesare logica va avansa la următoarea perioadă atunci când toate celelalte unități au terminat procesarea pasului curent.
- *Conservative*: Scopul acestei abordări îl reprezintă prevenirea erorilor de cauzalitate. Aste înseamnă ca, înainte de a avansa pa procesarea evenimentului cu ștampila de timp t, unitatea de procesare trebuie sa decidă daca evenimentul este „sigur" sau nu, adică se garantează faptul că în viitor nu o să apară un alt eveniment cu ștampila de timp inferioară.
- *Optimistice*: În acest caz unitățile de procesare pot sa încalce constrângerea de cauzalitate și sa execute evenimentele în ordinea primirii lor fără a încerca să prezică primirea în viitor a unui eveniment cu ștampila de timp inferioară. În cazul în care se întâmplă acest lucru atunci unitatea de procesare își va modifica starea curentă cu o valoare anterioară, considerată corectă, și va propaga această schimbare către celelalte unități de procesare interesate.

Toate aceste abordări au fost studiate în detaliu și s-a concluzionat ca performanta algoritmilor de sincronizare depinde de foarte mulți factori, cum ar fi: modelul simulării, mediul de execuție, scenariile specifice etc. Prognozarea performantei unui simulator paralel/distribuit se dovedește a fi foarte dificila deoarece depinde de o serie de factori ce nu pot fi definiți static, ei fiind dependenți de condițiile de execuție.

3. Structura sistemului de simulare

3.1. **Prezentare generala**

Scopul principal al sistemului propus este acela de a permite realizarea unui simulator pentru o instalație industriala folosind puterea de calcul a mai multor procesoare aflate pe aceiași mașină fizică sau chiar pe mașini diferite interconectate printr-un mediu de rețea. Pentru a putea atinge acest scop este util sa se realizeze o împărțire a instalației inițiale în secțiuni și realizarea de simulatoare pentru fiecare în parte și, la final, interconectarea acestora astfel încât sa se obțină aplicația dorită.

Aceasta abordare are avantajul simplificării procesului de dezvoltare deoarece componentele unei instalații vor avea simulatoare mult mai simple. În plus, daca într-o instalație apar mai multe componente identice ca funcționare (de exemplu mai multe pompe de același tip) atunci trebuie realizat un singur simulator pentru componenta respectiva iar în cadrul produsului final se vor folosi instanțe diferite ale acestuia, în funcție de necesități.

Un alt avantaj al acestei abordări este reprezentat de posibilitatea de a dezvolta diferitele simulatoare elementare în paralel datorita faptului ca aceste componente nu au o dependenta directa între ele. Un simulator pentru o secțiune, pe care îl vom numi simulator elementar(SE), va primi o serie de date din exterior, de la alte SE și, pe baza lor, își va determina starea curentă și/sau va genera un nou set de date pe care îl va trimite către alte SE. Prin analiza acestui comportament se ajunge la concluzia ca un simulator elementar este în esență un sistem dinamic.

Trebuie stabilită o modalitate de comunicare între SE. Comunicația directa între SE este cea mai rapida varianta dar contravine modelului general al simulatorului. Astfel modalitatea de interconectare nu trebuie sa depindă cu nimic de SE implicate. Cu alte cuvinte sistemul de interconectare trebuie sa poată fi folosit fără modificări indiferent de instalația de trebuie simulata. Mai mult trebuie avuta în vedere și situația în care în comunicație intervin mai multe componente sursa și/sau mai multe componente destinație iar acest lucru conduce la necesitatea de a introduce o noua componenta software între SE, componenta ce se ocupa doar cu transferul de date și care poarta numele de canal de comunicație(CC). CO sunt astfel componente software ce lucrează pe modelul producător-consumator. Ele permit conectarea unui număr nelimitat de producători și a unui număr nelimitat de consumatori și au rolul și obligația de a trimite un mesaj de la un producător către absolut toți consumatorii. Legătura între SE și CC se realizează prin intermediul unor componente software numite conectori.

Rezumând informațiile enunțate anterior putem spune ca în sistemul propus simulatorul pentru o instalație industriala reprezintă o reuniune de componente software a căror activitate este independenta dar care comunica unele cu altele și care pot fi plasate pe o singură mașină sau pe mașini diferite. Plasarea tuturor componentelor pe o singura mașină are costuri mai mici dar, în funcție de complexitatea instalației, poate fi ineficienta. Se poate constata ca puterea de calcul oferita este mai mica decât valoarea necesara pentru buna funcționare a sistemului iar acest lucru conduce la necesitatea folosirii mai multor mașini interconectate între ele. Se pune acum problema separării componentelor și împărțirii lor le mai multe mașini fizice. Datorita modului în care este implementat sistemul, aceasta separare se poate face la nivelul interconectării simulatoarelor elementare cu canalele

de comunicație și astfel pe fiecare mașina vor rula un număr de simulatoare elementare și/sau un număr de canale de comunicație.

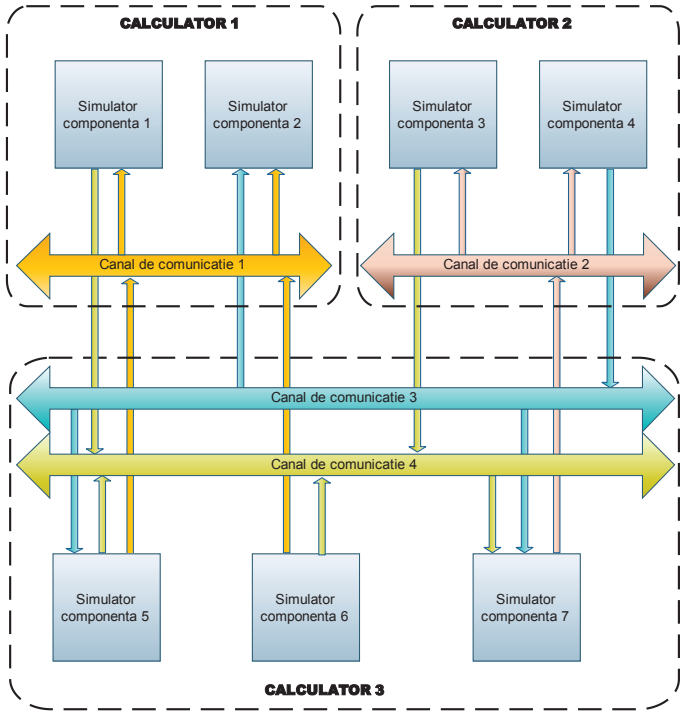

Figura 3.1 Simulator implementat pe trei mașini

Numărul simulatoarelor elementare ce intervin în realizarea arhitecturii sistemului este o variabila ce nu poate fi determinata cu un algoritm strict. El depinde de mai multe elemente, cum ar fi: complexitatea instalației ce trebuie simulata, eventualele simulatoare elementare deja implementate, priceperea persoanei responsabile cu împărțirea simulatorului în componente etc. Sistemul propus nu oferă o modalitate automata sau un algoritm pentru determinarea numărului de simulatoare elementare ci lașa acest aspect la alegerea responsabilului cu arhitectura simulatorului. Trebuie însă ținut cont de faptul ca un număr mare de componente simple poate avea avantajul timpului de implementare dar poate creste timpul de execuție datorita timpului necesar pentru transportul valorilor variabilelor de intrare și de ieșire, în timp ce un număr mic de componente are avantajul timpului de execuție mic, dar dezavantajul timpului de implementare mare dar și un grad scăzut de reutilizare a componentelor.

3.2. **Un algoritm pentru reducerea numărului de canale de comunicație**

Numărul canalelor de comunicație este în strânsă legătură cu numărul simulatoarelor elementare, mai exact cu numărul variabilelor schimbate între acestea, și poate varia într-un interval bine determinat, după cum urmează:

- Numărul minim de canale de comunicație în cadrul unei scheme ce conține cel puțin doua simulatoare elementare este de 1. Sistemul permite ca toate simulatoarele sa trimită valorile variabilelor de ieșire pe un singur canal și sa citească valorile variabilelor de intrare tot de pe acesta.
- Numărul maxim de canale de comunicație este egal cu numărul corespondentelor variabila de ieșire – variabila de intrare din schema.

Numărul optim de canale de comunicație reprezintă o valoare situata între cele doua numere prezentate mai sus și se poate obține folosind următorul algoritm:

1. Se reprezintă structura simulatorului sub forma unui graf orientat în care fiecare nod este un simulator elementar și fiecare muchie reprezintă o valoare schimbata între doua componente. Numărul muchiilor este în acest moment numărul maxim de canale de comunicație necesar pentru implementarea sistemului,
2. Se comasează muchiile care au aceiaşi sursa și aceiaşi valoare rezultând legături cu o sursa și mai multe destinații,
3. Se comasează legăturile cu o singura sursa și cu aceleași destinații rezultând legături cu mai multe surse și mai multe destinații. Numărul acestor legături este numărul optim de canale de comunicație.

3.3. Comunicația cu exteriorul

Un aspect important al sistemului îl constituie modalitatea de comunicare cu mediul exterior. Aceasta se împarte în doua componente:

- Comunicația cu echipamente reale
- Comunicația cu utilizatorii (interfața grafica)

3.3.1. Comunicația cu echipamente reale

Una din utilizările simulatoarelor este aceea de a testa buna funcționare a unor echipamente reale. Este o practica des întâlnită ca, atunci când se dezvolta echipamente hardware folosite pentru controlul și comanda unor instalații industriale, acestea sa fie testate intensiv folosind simulatoare înainte de a fi folosite împreună cu echipamentele reale, motivul principal fiind legat de riscul introdus de folosirea unor elemente aflate în faza de dezvoltare împreună cu instalații scumpe aflate deja în producție.

Sistemul propus, chiar daca nu oferă un suport evident, permite interfațarea cu echipamente reale. Astfel, pornind de la schema globala a simulatorului, se poate observa faptul ca un simulator elementar sau un ansamblu de simulatoare elementare poate fi înlocuit cu o componenta reala, singura modificare necesara fiind realizarea unui modul software pentru interfațare cu sistemul. O posibila varianta o poate reprezenta realizarea unei componente Java ce respecta, formatul simulatorului elementar și care apelează prin intermediul JNA o biblioteca nativa ce realizează conectarea efectiva cu echipamentul fizic.

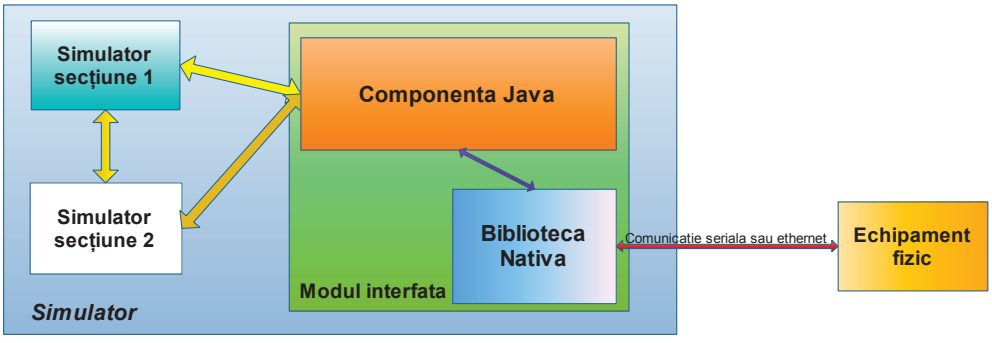

Figura 3.2 Comunicația cu echipamentele reale

3.3.2. Comunicația cu utilizatorii (interfața grafica)

Comunicația cu utilizatorii este modalitatea prin care simulatorul poate sa ofere operatorilor săi informații despre funcționare dar și posibilitatea sa intervină pentru a modifica comportamentul acestuia. Sistemul nu impune o interfața grafica, el putând rula fără probleme și fără aceasta, însă, datorita organizării sale, permite introducerea unei componente cu rol de interfața grafica. Practic interfața grafica poate fi considerata și ea un sistem dinamic ce conține variabile de intrare, variabile de ieșire, parametrii și variabile de stare, singura diferența fiind modul de interpretare al variabilelor de intrare și modul de generare al variabilelor de ieșire. Astfel, variabilele de intrare recepționate sunt procesate și afișate pe ecran într-un mod ce are semnificație pentru utilizatori. Aceștia, pe baza informațiilor afișate, pot sa ia anumite decizii și sa genereze valori pentru variabilele de ieșire. Putem spunem, pe baza analizei de mai sus, ca interfața grafica reprezintă o componenta a sistemului construita pe scheletul unui simulator elementar și având un comportament similar acestora. Se disting, în cazul acestei componente, doua aspecte:

- aspectul grafic – se referă la modul în care este organizată interfața grafică.
- aspectul funcțional – se referă la modul în care se realizează comunicația cu celelalte componente software ale sistemului.

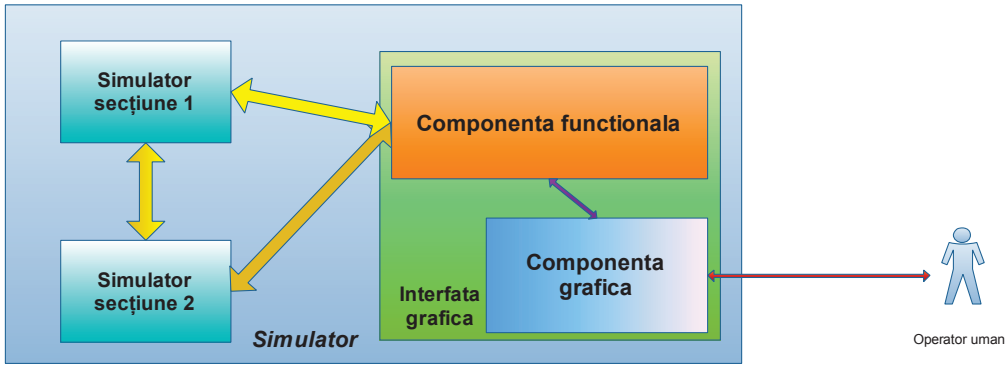

Figura 3.3 Componenta interfața grafica în cadrul simulatorului

3.4. Organizarea simulării

În acest moment sistemul nu impune cu nimic modul în care sunt împărțite componentele constituente, acesta fiind lăsat la alegerea persoanei ce se ocupa cu configurarea aplicației, și anume administratorul sistemului. Acesta poate încerca diferite împărțiri și va avea astfel posibilitatea sa aleagă cea mai buna varianta posibila. După ce a împărțit fizic componentele, adică le-a copiat pe mașinile implicate în realizarea simulatorului, administratorul va edita o serie de fișiere de configurare pentru fiecare mașină în parte creând astfel mai multe aplicații aparent independente între ele. Apoi va porni fiecare simulator în parte iar acestea vor realiza interconectarea între ele și apoi vor reproduce funcționarea dorita. Atunci când se dorește încheierea simulării este semnalat acest lucru la nivelul uneia dintre aplicații iar aceasta se va ocupa cu informarea restului componentelor despre cererea de terminare a execuției.

Pentru a evidenția modul de funcționare al sistemului trebuie detaliata modalitatea de configurare și funcționare a unei componente a sistemului care rulează pe o mașină și pe care o voi denumi în continuare aplicație simulator. O aplicație simulator este o aplicație Java ce rulează pe o mașină fizică și ce conține cel puțin o componenta a sistemului (fie simulator elementar, fie canal de comunicație). Schema bloc a aplicației simulator este prezentata în Figura 3.4.

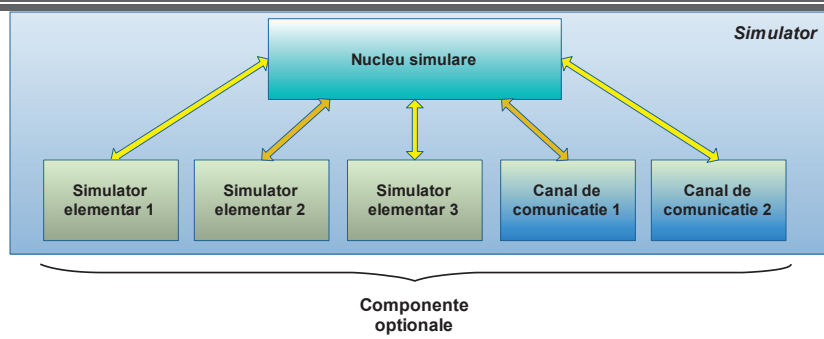

Figura 3.4 Schema bloc simulator

Nucleul de simulare reprezintă singura componenta obligatorie a simulatorului şi ea oferă cadrul necesar pentru demararea, configurarea şi buna funcţionare a simulatorului. Din punct de vedere structural aceasta componenta conţine doua subsisteme importante: gestiunea configuraţiei, responsabil cu citirea şi validarea fişierului de configuraţie al simulării şi gestiunea modulelor opţionale responsabil cu validarea şi iniţializarea canalelor de comunicaţie şi/sau a simulatoarelor elementare folosite în cadrul simulării pe maşina curentă.

Pe lângă componenta obligatorie, aplicaţia simulator poate conţine şi componente opţionale, a căror prezenta este definita prin intermediul configuraţiei şi verificata doar în momentul execuţiei. Astfel, nu exista o dependenta directa între aplicaţia simulator şi modulele care implementează simulatoare elementare sau canale de comunicaţie.

3.4.1. Fişierul de configuraţie

Fişierul de configuraţie este un fişier text, în format XML şi este folosit pentru determinarea modulelor (simulatoare elementare şi canale de comunicaţie) ce trebuie create pe maşina locala precum şi modul în care acestea comunica între ele pentru realizarea simulatorului dorit. Fişierul de configuraţie este împărţit în trei secţiuni opţionale şi anume definiţii, canale de comunicaţie şi simulatoare elementare, fiecare cu o semnificaţie aparte:

- **Secţiunea definiţii.** Aceasta secţiune este folosita intern de modulul de citire şi validare a configuraţiei pentru scrierea aceleiaşi valori pentru mai mulţi parametrii sau variabile ale sistemului.
- **Secţiunea canale de comunicaţie.** Aceasta secţiune este folosita pentru definirea canalelor de comunicaţie ce trebuie create pe maşina curentă sau sunt folosite direct de simulatoare elementare existente pe maşina curentă.
- **Secţiunea simulatoare elementare.** Aceasta secţiune este folosita pentru configurarea simulatoarelor elementare ce rulează pe maşina curentă precum şi, daca este cazul, a componentei ce gestionează interfaţa grafica şi care respecta şi extinde funcţionarea unui simulator elementar.

Descrierea simulatorului prin intermediul unui fişier de configuraţie reprezintă unul din avantajele sistemului curent deoarece în momentul execuţiei se pot verifica diferite arhitecturi pana când se vor obţine rezultatele dorite.

3.4.2. Funcţionarea aplicaţiei simulator

Din punct de vedere funcţional execuţia aplicaţiei simulator urmăreşte un număr de paşi bine determinaţi şi anume:

- **Demararea aplicației simulator.** Acesta este primul pas și corespunde momentului în care este pornita aplicația și este singurul pas în care aceasta solicita intervenția unui operator uman. La pornire aplicației i se oferă o serie de parametrii (prin intermediul unui fișier de proprietăți), aceștia fiind necesari pentru identificarea tuturor resurselor necesare pentru o buna funcționare.
- **Configurarea simulării.** Configurarea simulării presupune citirea și validarea fișierului de configurare.
- **Inițializarea simulării.** Etapa de inițializare a simulării presupune crearea canalelor de comunicație și a simulatoarelor elementare implicate în simulare precum și inițializarea lor cu valorile specificate în fișierul de configurare
- **Pornirea simulării.** Pornirea simulării presupune trei etape, și anume demararea canalelor de comunicație, conectarea simulatoarelor elementare la canalele de comunicație și demararea simulatoarelor elementare. După finalizarea ultimei etape simularea este considerată "în execuție".
- **Oprirea simulării.** Oprirea simulării se execută în momentul recepționarii unei comenzi sub forma unui mesaj UDP multicast, broadcast sau unicast, după caz. Când aceasta este recepționată se opresc simulatoarele elementare, se deconectează simulatoarele elementare de la canalele de comunicație și se opresc canalele de comunicație și se încheie execuția.

4. Simulatoare elementare

4.1. Prezentare generală

Simulatoarele elementare reprezintă nucleul întregului sistem. Ele sunt componentele care formează instalaţia industriala care trebuie simulata şi deci vor fi specifice fiecărei instalații. În esenţa fiecare simulator elementar poate fi considerat tot o instalație dar cu dimensiuni mai mici decât instalaţia iniţiala. Ideal ar fi ca toate aceste simulatoare elementare sa poată fi simplificate la maxim, dar se poate întâmpla ca în anumite situaţii aceasta simplificare sa conducă la apariţia unor complicaţii la interconectarea lor şi implicit la întârzieri în execuție. Ceea ce se poate spune despre toate simulatoarele elementare este faptul ca, indiferent de gradul lor de complexitate, ele vor fi sisteme dinamice şi se vor supune regulilor aplicate acestora.

Din punct de vedere al funcţionarii orice simulator elementar trebuie sa efectueze următorii paşi:

4. Iniţializare. Se citesc valorile iniţiale pentru parametrii şi variabilele de stare;
5. Conectare. Simulatorul elementar se conectează la canalele de comunicație folosite pentru citirea valorilor variabilelor de intrare și la canalele de comunicație folosite pentru trimiterea noilor valori ale variabilelor de ieșire;
6. Execuție. Se calculează și se trimit noile valori ale variabilelor de ieșire.
7. Oprire. Se oprește execuția funcționalității specifice și se întrerupe legătura cu canalele de comunicație.

Analizând algoritmul de mai sus se poate observa foarte uşor că paşii 1, 2, şi 4 sunt comuni pentru toate simulatoarele elementare, în timp ce pasul 3 este specific pentru fiecare componenta în parte. Pe baza acestei analize rezultă concluzia că orice simulator elementar este format din doua componente:

- o componenta independenta de funcția propriu-zisa a simulatorului ce se ocupa cu inițializarea parametrilor și variabilelor de stare și de ieșire, cu scrierea valorilor noi ale variabilelor de intrare și cu transmiterea valorilor variabilelor de ieșire;
- o componenta specifica pentru fiecare simulator în parte ce se ocupa cu validarea valorilor variabilelor de intrare și, cel mai important, cu calculul noilor valori pentru variabilele de stare și de ieșire.

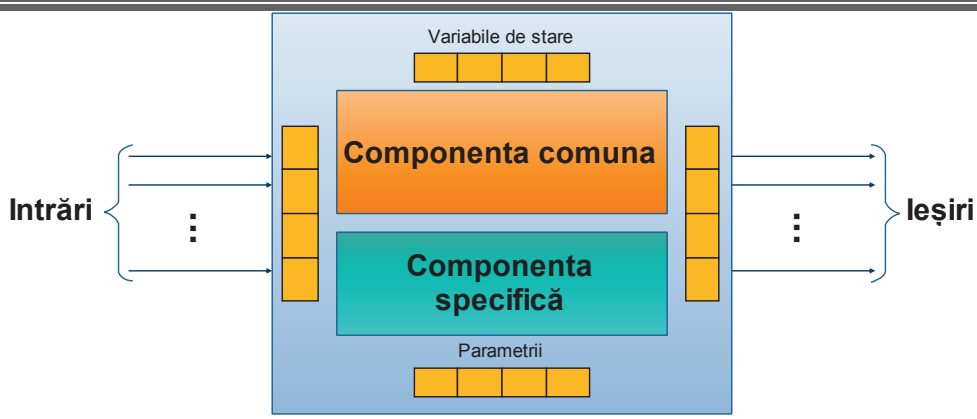

Figura 4.1 Simulator elementar format din două componente

Sistemul oferă o implementare a elementelor comune pentru toate simulatoarele sub forma unei componente numita simulator elementar abstract.

4.2. **Simulatorul elementar abstract**

Aceasta componenta nu poate funcționa de sine stătător ci doar oferă o implementare pentru toate simulatoarele elementare, scopul său principal fiind oferirea suportului necesar pentru dezvoltarea rapidă a componentelor. Astfel, prin folosirea acestei componente, un dezvoltator nu trebuie sa se preocupe de decât de partea funcțională a simulatorului pe care dorește să-l realizeze. Simulatorul elementar abstract se ocupa integral de inițializarea, conectarea și oprirea simulatorului elementar și parțial de faza de execuție.

Etapa de execuție definește modalitatea prin care se generează noile valori ale variabilelor de ieșire pe baza variabilelor de intrare a parametrilor și a variabilelor de stare. Nu se poate defini un mod unitar pentru generarea valorilor variabilelor de ieșire dar se pot observa anumite modele. Astfel, în majoritatea cazurilor, schimbarea valorii unei variabile de ieșire depinde direct de schimbarea valorii unei variabile de intrare. Aceasta procedura poate fi descrisa de algoritmul următor:

8. Simulatorul așteaptă apariția unor valori pentru una sau mai multe variabile de intrare;
9. Se citesc noile valori ale variabilelor de intrare și se salvează intern pentru folosirea în cadrul calculelor specifice;
10. Daca este necesar, se calculează noile valori pentru variabilele de stare și pentru variabilele de ieșire;
11. Se transmit, daca este cazul, valorile variabilelor de ieșire și se revine la pasul 1.

Din cei patru pași enunțați mai sus 1, 2 și 4 sunt independenți de funcționalitatea dorita pentru o componenta și pot fi implementați în mod unitar la nivelul simulatorului elementar abstract, în timp ce pasul trei este specific fiecărei componente și nu poate fi realizat în mod automat.

4.2.1. Algoritmul de serializare/deserializare a datelor

Operațiile de serializare și respectiv deserializare sunt foarte importante în funcționarea simulatorului elementar abstract. Necesitatea acestor operații provine din faptul ca la nivelul canalelor de comunicație datele nu au o semnificație concreta ci sunt văzute ca șiruri de octeți, în timp ce la nivelul simulatoarelor elementare ele reprezintă valori concrete ale unor variabile bine definite. Java oferă o modalitate automata de serializare și deserializare, dar algoritmul folosit adaugă foarte multe

informații suplimentare fata de valorile variabilelor. Din acest motiv a fost dezvoltat un algoritm de serializare/deserializare simplu ce permite crearea unor pachete cu dimensiuni mici.

Conform algoritmului propus pachetele schimbate prin intermediul canalelor de comunicație sunt șiruri de octeți și conțin doua secțiuni:

- Secțiunea header – conține informații pentru identificarea componentei sursa a pachetului.
- Secțiunea date – conține valorile variabilelor trimise în cadrul pachetului.

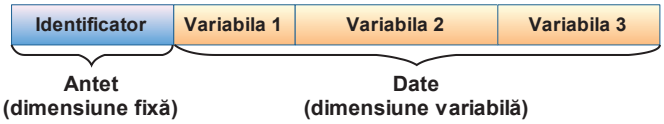

Figura 4.2 Structura pachet

Toate variabilele transmise sunt serializate respectând formatul rețea.

4.2.2. Contribuții privind implementarea simulatorului elementar abstract

Din punct de vedere al implementării simulatorul elementar abstract reprezintă o clasa Java. O reprezentare schematica este prezentata în Figura 4.3.

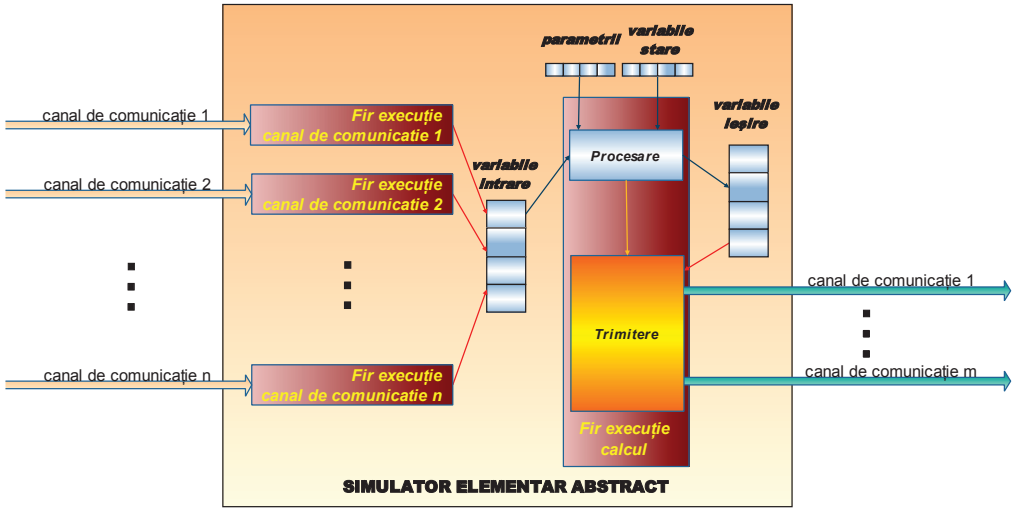

Figura 4.3 Simulatorul elementar abstract

La nivelul simulatorului elementar abstract nu sunt definite nici un fel de variabile de intrare, de stare, de ieșire sau parametrii, ei fiind specifici pentru fiecare simulator ce folosește aceasta componenta, însă este definita o modalitate de identificare și accesare a acestora. Astfel, în momentul creări unui simulator ce extinde componenta abstracta, dezvoltatorul va declara membrii Java pentru fiecare variabila sau parametru utilizat pe care îi va identifica folosind adnotările Java (Java Annotations).

Pentru a putea primi valorile variabilelor de intrare de pe mai multe canale de comunicație în paralel, simulatorul elementar abstract folosește fire de execuție. Astfel, pentru fiecare canal de comunicație prin intermediul căruia se citesc valori ale variabilelor de intrare, se creează un fir de execuție separat. Acestea sunt blocate în așteptarea unor date și încep procesarea numai daca se primesc noi valori ale variabilelor de intrare prin intermediul canalului de comunicație corespondent.

Algoritmul după care funcționează fiecare astfel de fir de execuție este:

```
cat timp firul de execuţie este pornit
|  aşteaptă un pachet de date de la canalul de comunicaţie
|  dacă conexiunea este întreruptă
|  |  opreşte firul de execuţie
|  └#
|  altfel
|  |  extrage identificatorul sursei datelor
|  |  daca identificatorul este acceptat
|  |  |  extrage datele din pachet conform configuraţiei
|  |  |  informează firul de execuţie ce se ocupa de recalculare
└# └# └#
```

După recepționarea noilor valori ale variabilelor de intrare prin intermediul unui fir de execuție trebuie sa se recalculeze noile valori pentru variabilele de stare și de ieșire. Simulatorul elementar abstract introduce noțiunea de fluxuri de date. Un flux de date definește o legătura între variabilele de intrare și variabilele de ieșire. Astfel, daca o variabila de intrare și o variabila de ieșire aparțin aceluiași flux de date, atunci recepționarea unei noi valori pentru prima va conduce la emiterea valorii celei de-a doua, chiar daca aceasta nu a suferit vre-o modificare. Altfel, daca nu se recepționează nici o valoare pentru variabilele de intrare ce aparțin unui flux de date atunci valorile variabilelor de ieșire din același flux nu vor fi niciodată transmise chiar daca sunt modificate în interiorul componentei.

Pentru a calcula valorile variabilelor de stare și a variabilelor de ieșire, simulatorul elementar abstract folosește un fir de execuție dedicat. Acesta este pornit odată cu pornirea simulatorului și se blochează în așteptarea unor cereri de procesare a datelor de intrare. În momentul în care o astfel de cerere este recepționată valorile variabilelor de intrare a căror recepție a declanșat recalcularea vor fi scrise în câmpurile interne ale simulatorului elementar. Scrierea valorilor recepționate în variabilele corespondente se face folosind „Reflection”.

După ce operația de scriere în câmpurile interne a valorilor variabilelor de intrare este încheiată se verifică dacă este necesară o recalculare a valorilor variabilelor de stare și de ieșire. În cazul în care cel puțin una din variabilele de intrare a cărei valoare a fost recepționată este marcată ca aparținând unui flux de date atunci se realizează procesarea cu ajutorul unei funcții implementate la nivelul componentei ce extinde simulatorul elementar abstract. În funcție de rezultatul procesării valorile variabilelor de ieșire corespunzătoare fluxurilor de date ce au declanșat procesarea sunt transmise sau nu către canalele de comunicație stabilite conform configurației. Pe scurt, algoritmul implementat de firul de execuție folosit pentru recalcularea valorilor variabilelor de stare și de ieșire este:

```
cat timp simulatorul este pornit
|  aşteaptă o cerere de recalculare
|  salvează valorile variabilelor de intrare in câmpurile corespondente
|  dacă este necesară recalcularea
|  |  apelează funcţia externa simulatorului elementar abstract
|  |  daca se solicita trimiterea către destinaţii a variabilelor de ieşire
|  |  |  citeşte valorile variabilelor de ieşire
|  |  |  trimite valorile citite către toţi conectorii de ieşire
└# └# └#
```

Citirea valorilor variabilelor de ieșire se face folosind tot procedeul „Reflection”.

4.3. **Analiza eficientei simulatorului elementar abstract**

Implementarea abstracta de simulator elementar oferita de sistemul actual îi permite unui dezvoltator sa realizeze diverse componente foarte rapid, fiind necesara doar implementarea a maxim patru metode, numărul acesta depinzând de modalitatea în care se dorește actualizarea valorilor pentru variabilele de intrare și citirea valorilor variabilelor de ieșire (prin Reflection sau direct, prin

suprascrierea metodelor „`setInputVariableValue`" și respectiv „`getOutputVariableValue`" în componenta). Este important de analizat daca aceasta abordare care permite implementarea rapida a componentelor sistemului este și eficienta din punct de vedere al timpului de execuție ce poate fi împărțit în doua componente: timpul din momentul în care datele de intrare părăsesc conectorul corespunzător și momentul în care se apelează metoda interna de procesare a datelor și timpul de când se încheie procesarea și pana când datele ajung la nivelul conectorului de ieșire. În Figura 4.4 este prezentat montajul folosit pentru calcularea timpilor precizați mai sus. Prin adunarea celor doua valori calculate mai sus ($T_{intrare}$ și $T_{ieșire}$) se obține întârzierea totala introdusa de folosirea SEA.

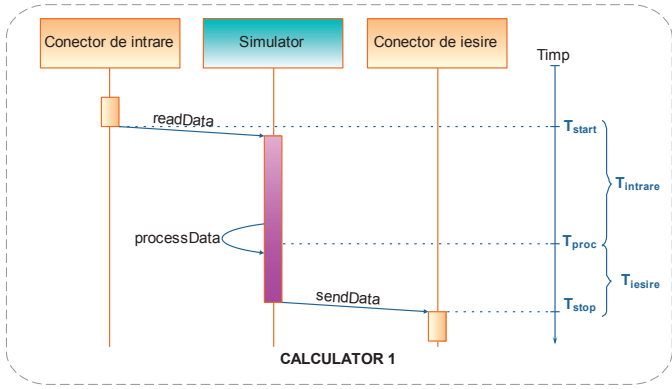

Figura 4.4 Montaj evaluare timp de execuție simulator elementar abstract

Pentru a măsura timpii de întârziere enunțați mai sus s-au realizat cate șapte teste pentru cazul în care valorile sunt setate și citite prin Reflection și respectiv șapte teste pentru cazul în care valorile variabilelor sunt scrise și citite direct. Fiecare test a presupus scrierea și citirea unei variabile de tip boolean, byte, short, int, long, float și respectiv double și a fost repetat de 10000 de ori. În Tabelul 4.1, Tabelul 4.2 și respectiv Tabelul 4.3 sunt sintetizate valorile minime, maxime și medii ale timpului de intrare, timpului de ieșire și respectiv timpului total în cele două cazuri studiate.

Modalitate scriere/citire valori	**Tip de date**	**Timp mediu (ns)**	**Timp maxim (ns)**	**Timp minim (ns)**	**Mai mic decât dublul valorii medii (%)**
Reflection	Boolean	14883	182332	6649	95.91%
	Byte	14334	169033	5249	96.47%
	Short	14959	141036	5599	96.12%
	Integer	15151	150485	5599	96.36%
	Long	14997	155734	5949	97.07%
	Float	14796	134387	5949	96.73%
	Double	15315	97291	5949	96.23%
Direct	Boolean	13613	137527	5600	97.04%
	Byte	13646	165183	5249	96.53%
	Short	14281	180932	5249	96.67%
	Integer	13814	191430	5249	96.73%
	Long	14794	206830	5600	96.51%
	Float	14154	145235	5949	96.74%
	Double	14608	144535	5599	96.12%

Tabelul 4.1 Valori statistice pentru timpul de intrare

Modalitate scriere/citire valori	Tip de date	Timp mediu (ns)	Timp maxim (ns)	Timp minim (ns)	Mai mic decât dublul valorii medii (%)
Reflection	Boolean	3979	38496	1400	93.67%
	Byte	3975	37096	1049	93.16%
	Short	4122	38146	1050	92.62%
	Integer	4153	39546	1399	94.07%
	Long	4346	38146	1050	92.01%
	Float	4219	38847	1050	93.86%
	Double	4330	38146	1050	92.11%
Direct	Boolean	4208	52495	1399	93.38%
	Byte	4075	64043	1399	93.32%
	Short	4233	61594	1400	93.71%
	Integer	4272	65444	1399	94.43%
	Long	4606	54945	1400	91.50%
	Float	4449	50395	1749	94.13%
	Double	4555	52145	1399	91.69%

Tabelul 4.2 Valori statistice pentru timpul de ieşire

Modalitate scriere/citire valori	Tip de date	Timp mediu (ns)	Timp maxim (ns)	Timp minim (ns)	Mai mic decât dublul valorii medii (%)
Reflection	Boolean	18862	187231	8399	94.45%
	Byte	18308	179882	6299	94.49%
	Short	19081	179182	6999	94.39%
	Integer	19305	156784	6999	94.84%
	Long	19343	161333	7349	93.66%
	Float	19015	139636	7349	94.23%
	Double	19645	133687	7349	92.71%
Direct	Boolean	17820	164483	7349	95.58%
	Byte	17721	172182	6649	95.62%
	Short	18514	187931	6999	95.59%
	Integer	18086	201929	6650	95.24%
	Long	19400	214179	7349	94.59%
	Float	18603	184431	7699	94.97%
	Double	19163	148035	6999	93.30%

Tabelul 4.3 Valori statistice pentru timpul total

Primele concluzii pe care le putem extrage din aceste tabele sunt:

- Simulatorul elementar abstract se comporta asemănător din punct de vedere al timpului folosit, indiferent de modalitatea aleasă pentru scrierea sau citirea valorii variabilelor,
- În peste 92% din cazurile testate timpul total necesar este mai mic decât 40µs
- În cel mai rău caz intarzierea introdusă de SEA este de 214µs,
- Nu se observa o dependenta directa între dimensiunea în octeţi a variabilei şi timpul măsurat
- Valorile medii calculate în cazul în care se foloseşte Reflection sunt cu maxim 1,5µs mai mari decât cazul în care se foloseşte metoda suprascrierii directe a funcţiilor.

În Tabelul 4.4, Tabelul 4.5 și Tabelul 4.6 sunt prezentate valorile medii de calculate pentru primele 1500 de mesaje comparativ cu cele calculate pentru următoarele 8500 de mesaje.

Modalitate scriere/citire valori	**Tip de date**	**Timp mediu primele 1500 de mesaje (ns)**	**Timp mediu următoarele 8500 de mesaje (ns)**	**Diferența (%)**	**Mai mic decât dublul valorii medii pentru următoarele 8500 de mesaje (%)**
Reflection	Boolean	22022	13624	38.14%	95.99%
	Byte	21438	13080	38.99%	96.66%
	Short	22931	13552	40.90%	96.29%
	Integer	22026	13938	36.72%	95.36%
	Long	23833	13438	43.62%	97.33%
	Float	22408	13452	39.97%	96.60%
	Double	24310	13728	43.53%	96.28%
Direct	Boolean	19310	12607	34.71%	97.39%
	Byte	18576	12776	31.22%	96.55%
	Short	20341	13212	35.05%	96.89%
	Integer	18814	12931	31.27%	96.73%
	Long	21791	13559	37.78%	96.32%
	Float	20213	13085	35.26%	96.95%
	Double	21754	13347	38.65%	96.04%

Tabelul 4.4 Analiza comparativa a timpului de intrare

Modalitate scriere/citire valori	**Tip de date**	**Timp mediu primele 1500 de mesaje (ns)**	**Timp mediu următoarele 8500 de mesaje (ns)**	**Diferența (%)**	**Mai mic decât dublul valorii medii pentru următoarele 8500 de mesaje (%)**
Reflection	Boolean	9340	3032	67.53%	99.61%
	Byte	9370	3023	67.74%	99.54%
	Short	9619	3152	67.23%	99.67%
	Integer	9336	3239	65.31%	99.64%
	Long	10256	3303	67.79%	99.59%
	Float	9655	3260	66.23%	99.62%
	Double	10610	3222	69.63%	99.66%
Direct	Boolean	10181	3153	69.03%	96.99%
	Byte	9908	3046	69.26%	96.66%
	Short	10115	3194	68.42%	97.33%
	Integer	9901	3279	66.89%	96.92%
	Long	11221	3439	69.35%	98.07%
	Float	10229	3429	66.48%	97.46%
	Double	11533	3323	71.18%	96.95%

Tabelul 4.5 Analiza comparativa a timpului de ieşire

Modalitate scriere/citire valori	Tip de date	Timp mediu primele 1500 de mesaje (ns)	Timp mediu următoarele 8500 de mesaje (ns)	Diferența (%)	Mai mic decât dublul valorii medii pentru următoarele 8500 de mesaje (%)
Reflection	Boolean	31362	16656	46.89%	96.21%
	Byte	30808	16103	47.73%	96.67%
	Short	32550	16704	48.68%	96.39%
	Integer	31362	17177	45.23%	96.26%
	Long	34089	16741	50.89%	97.39%
	Float	32063	16712	47.88%	97.00%
	Double	34919	16950	51.46%	96.38%
Direct	Boolean	29491	15761	46.56%	97.42%
	Byte	28484	15822	44.45%	96.56%
	Short	30456	16406	46.13%	96.96%
	Integer	28716	16210	43.55%	96.78%
	Long	33011	16998	48.51%	96.46%
	Float	30442	16514	45.75%	96.95%
	Double	33286	16670	49.92%	96.13%

Tabelul 4.6 Analiza comparativa a timpului total

Prin aceasta analiza putem concluziona faptul ca, prin folosirea simulatorului elementar abstract ca suport pentru toate simulatoarele elementare simplifica la maxim dezvoltarea cu un cost total de timp de maxim 34 de microsecunde în peste 96% din cazuri.

4.4. **Simulatoare speciale**

Pentru a veni în ajutorul dezvoltatorilor sistemul pune la dispoziție doua componente construite pe arhitectura simulatorului elementar abstract, dar care nu funcționează în totalitate după modelul general implementat de acesta, și anume modificarea valorilor variabilelor de ieșire numai după modificarea a cel puțin o variabila de intrare. Cele doua componente sunt interfața grafica și generatorul de semnale de ceas și sunt prezentate în continuare.

4.4.1. Componenta interfața grafica

Componenta interfața grafica oferă o modalitate simpla prin intermediul căreia se pot obține comenzi de la un utilizator uman sau se pot afișa rezultatele simulării.

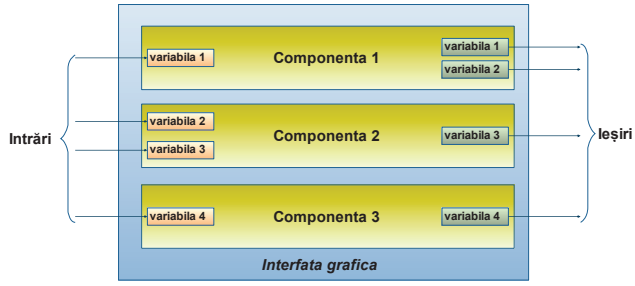

Figura 4.5 Diagrama interfața grafica

Din punct de vedere funcțional aceasta componenta se conectează la canalele de comunicație la fel ca orice simulator elementar, diferența importanta fiind aceea ca modificarea unei valori pentru o variabila de intrare nu înseamnă în mod automat recalcularea și mai ales transmiterea valorilor variabilelor de ieșire. Diagrama componentei interfața grafica este prezentata în Figura 4.5.

4.4.2. Componenta generatoare de semnale de ceas

Rolul componentei generatoare de semnale de ceas este transmiterea valorii unei variabile de ieșire la intervale de timp prestabilite prin intermediul fișierului de configurație. Componenta este realizata prin extinderea simulatorului elementar abstract și implementarea celor doua funcții obligatorii, conform structurii acestuia. În plus această componentă conține un fir de execuție a cărui durată de viață este controlată de valoarea variabilei de intrare și care va trimite periodic valoarea parametrului „`timeInterval`”. Diagrama acestei componente este prezentata în Figura 4.6.

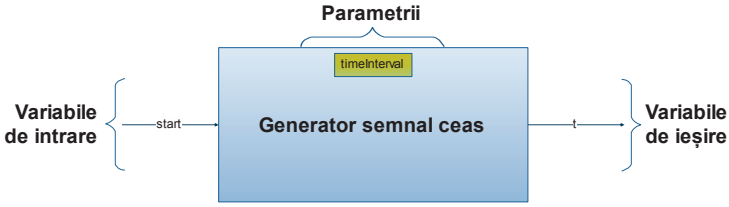

Figura 4.6 Diagrama Generator Semnal de Ceas

5. Canalele de comunicație

5.1. Prezentare generală

Canalele de comunicație reprezintă componentele sistemului de simulare responsabile cu transportul mărimilor generate de unul sau mai multe simulatoare elementare sursa (producători) către unul sau mai multe simulatoare elementare destinație (consumatori).

Canalele de comunicație transporta doar date abstracte,fără o semnificație, reprezentate sub forma unor șiruri de octeți. Ele nu realizează nici un fel de filtrare sau interpretare a datelor ci doar se ocupa cu transferul lor către toate destinațiile înregistrate. Este de datoria componentei destinație sa analizeze datele și sa determine daca îi sunt utile sau trebuie ignorate. Din acest motiv este important ca prin configurația sistemului sa se stabilească legăturile între componente în așa fel încât sa nu fie transferuri inutile sau măcar numărul lor sa fie foarte mic. Legătura între un canal de comunicație și un simulator se realizează prin intermediul unei componente software numită conector.

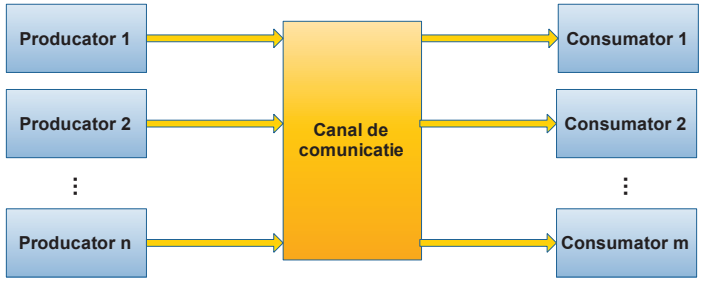

Figura 5.1 Canal de comunicație

În funcție de modul în care sunt folosiți, conectorii se împart în doua categorii:

- Conectori de intrare – folosiți pentru a primi de la canalul de comunicație corespondent valorile unor variabile de intrare.
- Conectori de ieșire – folosiți pentru a trimite către canalul de comunicație corespondent valorile unor variabile de ieșire.

Canalele de comunicație se împart în 2 categorii:

12. canale de comunicație locala: sunt acele canale de comunicație care permit numai interconectarea unor entități ce rulează în cadrul aceleiași mașini virtuale Java.
13. canale de comunicație la distanta: sunt acele canale de comunicație care permit interconectarea unor entități ce se afla fie pe aceiași mașina, fie pe mașini diferite.

De fapt sistemul nu impune canalele de comunicație, ci doar definește un set de reguli ce trebuie respectate de acestea, după cum urmează:

- Un canal de comunicație trebuie sa ofere trei funcționalități principale: inițializare, pornire și oprire,
- Un canal de comunicație trebuie sa ofere cel puțin o implementare pentru un conector de intrare și cel puțin o implementare pentru un conector de ieșire.
- Un canal de comunicație trebui sa ofere conectorilor săi detaliile de conectare chiar și daca nu este pornit pe mașina curentă;
- Modificarea canalului de comunicație folosit în cadrul unei comunicații trebuie sa se realizeze doar prin modificarea fișierului de configurație.
- Un canal de comunicație trebuie sa garanteze trimiterea tuturor mesajelor recepționate de la elementele producător către elementele consumator,
- Un canal de comunicație trebuie sa garanteze faptul ca ordinea de transmitere a mesajelor de la o componenta producător este aceiași cu ordinea de recepționare de către consumatori.
- Cat timp la nivelul canalului de comunicație nu exista date disponibile, toți conectorii de intrare se blochează în așteptarea informațiilor dorite.

Pentru a veni în sprijinul dezvoltatorilor, sistemul actual oferă patru implementări pentru canalele de comunicație, toate respectând cerințele de mai sus : doua implementări pentru canale de comunicație locala (capabile sa transfere date doar între componente aflate pe aceiași mașină) și doua implementări pentru canale de comunicație la distanta (a căror funcție principala este transferarea de date atât între doua componente aflate pe mașini diferite).

5.2. **Canale de comunicaţie locală**

5.2.1. Canal de comunicație bazat pe memorie partajată

Din punct de vedere al implementării, canalul de comunicație este reprezentat de o zona comuna de memorie în care producătorii scriu date iar consumatorii citesc date.

Atunci când un producător dorește sa scrie date, va verifica daca zona de memorie nu este deja ocupata, caz în care va fi nevoit sa aștepte eliberarea acesteia. Daca zona de memorie este libera, datele vor fi scrise acolo și apoi toți consumatorii vor fi informați asupra acestui lucru. Din acest moment consumatorii vor citi datele scrise și le vor procesa. Accesul lor la zona de memorie se face secvențial iar ultimul consumator care citește datele va marca zona ca fiind libera. Daca nu exista date în zona de memorie atunci un apel de citire va bloca firul de execuție curent în așteptarea unui mesaj ce poate fi procesat, respectându-se astfel cerințele generale ale canalelor de comunicație enunțate mai sus.

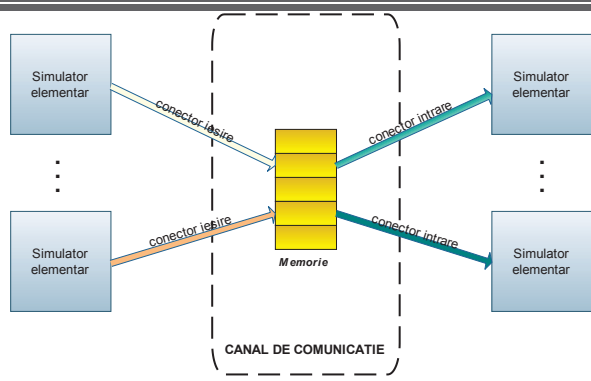

Figura 5.2 Canal de comunicație bazat pe memorie partajata

5.2.2. Canal de comunicație bazat pe cozi de mesaje

Acest canal de comunicație alocă pentru fiecare consumator conectat o coadă de mesaje cu dimensiune fixa și păstrează aceste obiecte într-o lista interna.

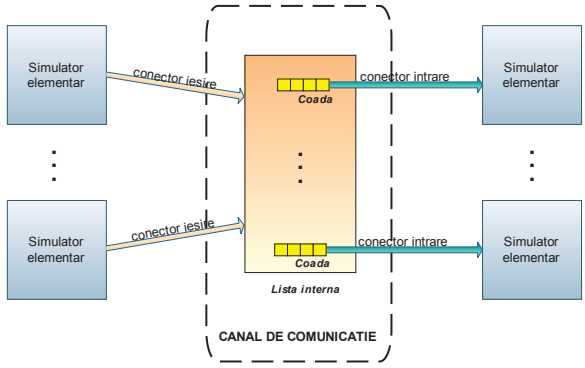

Figura 5.3 Canal de comunicație bazat pe cozi de mesaje

Atunci când un producător doreşte sa transmită un mesaj, el va obţine acces exclusiv asupra acestei liste şi va copia mesajul sau în fiecare din cozile disponibile. Diferenţa majora fata de canalul anterior o constituie faptul ca, o scriere nu este neapărat urmat de un număr de citiri egal cu numărul de consumatori şi astfel un consumator mai lent nu va influenta execuţia unor componente mai rapide. Pentru a se asigura accesul concurent la date aceasta componenta foloseşte cozi cu blocare.

5.2.3. Analiza eficientei canalelor de comunicaţie locale

În cazul canalelor locale cele doua componente (producătorul şi consumatorul) au fost poziţionate pe aceiaşi maşina fizica iar măsurarea s-a efectuat prin citirea ştampilei de timp din momentul imediat anterior transmiterii unui mesaj şi citirea ştampilei de timp din momentul imediat ulterior primirii mesajului.

Diferenţa dintre aceste ştampile de timp reprezintă o aproximare buna a timpului necesar pentru transmiterea unui mesaj prin intermediul canalului de comunicație.

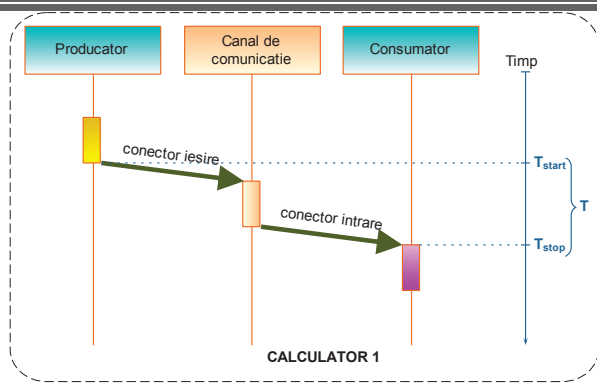

Figura 5.4 Măsurare timp de propagare canal de comunicație local

Valorile de timp minime, maxime și medii sunt prezentate în Tabelul 5.1.

Canal de comunicație	**Dimensiune date (octeți)**	**Timp mediu (ns)**	**Timp maxim (ns)**	**Timp minim (ns)**	**Mai mic decât dublul valorii medii (%)**
Memorie partajata	1	10395	114088	3499	99,43%
	2	11061	130887	3500	97,53%
	4	11265	247425	3499	97,74%
	8	10708	203679	3499	96,79%
	16	11135	254775	3500	97,54%
	32	11233	215228	3500	97,31%
Cozi de mesaje	1	12678	226077	3849	92,26%
	2	12306	301670	3849	93,24%
	4	12666	203330	3849	92,81%
	8	12315	226427	3849	92,67%
	16	12314	294321	3849	92,59%
	32	12829	230277	3849	93,34%

Tabelul 5.1 Statistici canale de comunicație locale (I)

În Tabelul 5.2 sunt prezentate comparativ valorile medii măsurate pentru primele 1500 de mesaje și cele măsurate pentru următoarele 8500 de mesaje

Canal de comunicație	**Dimensiune date (octeți)**	**Timp mediu primele 1500 de mesaje (ns)**	**Timp mediu următoarele 8500 de mesaje (ns)**	**Diferența (%)**	**Mai mic decât dublul valorii medii pentru următoarele 8500 de mesaje (%)**
Memorie partajata	1	12773	9976	21.90%	99.56%
	2	13288	10668	19.72%	97.84%
	4	13979	10786	22.85%	97.92%
	8	13508	10214	24.39%	97.53%
	16	13563	10706	21.06%	97.60%
	32	13552	10823	20.13%	97.15%
Cozi de mesaje	1	23536	10762	54.28%	97.27%
	2	22891	10438	54.40%	96.55%

	4	23634	10730	54.60%	96.94%
	8	23025	10425	54.72%	97.36%
	16	22620	10496	53.60%	97.29%
	32	23675	10915	53.89%	97.55%

Tabelul 5.2 Statistici canale de comunicație locale (II)

Se observa că, după primele 1500 de mesaje, timpul necesar pentru transmitere de la producător la consumator este în peste 96% din cazuri mai mic sau egal cu 22µs.

5.3. Canale de comunicație distribuite

5.3.1. Canal de comunicație bazat pe un protocol simplu peste TCP/IP

Acest canal de comunicație reprezintă o implementare simpla a interfețelor impuse de sistem, ce permite transportul mesajelor între doua sau mai multe mașini legate între ele prin intermediul unei rețele. După cum îi spune și numele acest canal de comunicație folosește transferul datelor prin socket stream.

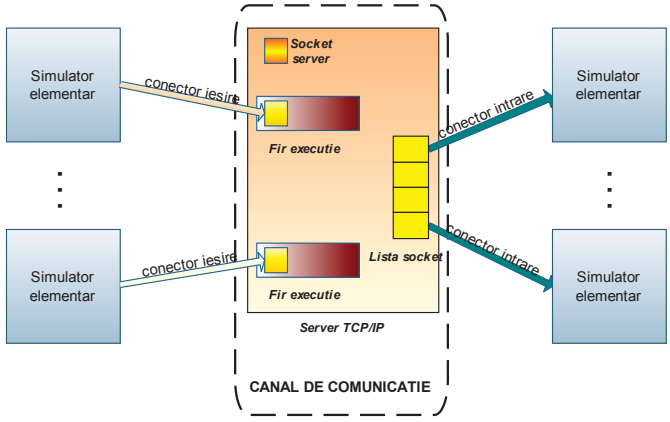

Figura 5.5. Canal de comunicație bazat pe un protocol simplu peste TCP/IP

În esență acest canal de comunicație pornește un fir de execuție ce conține un socket stream configurat sa aștepte cereri de conexiuni. Aceste cereri provin de la conectorii de intrare și respectiv de ieșire care astfel acționează ca niște clienți stream. În momentul în care un conector de intrare se conectează la canalul de comunicație se va crea un obiect socket ce va fi adăugat într-o lista interna și va rămâne acolo pana când conectorul respectiv se deconectează sau canalul de comunicație este oprit de utilizator.

Atunci când un conector de ieșire se conectează la canalul de comunicație se va crea tot un obiect socket dar acesta nu este adăugat în lista interna ci se va crea un nou fir de execuție care se va bloca în așteptarea unor date.

5.3.2. Canal de comunicație bazat pe JMS

Canalul de comunicație bazat pe JMS folosește implementarea ActiveMQ oferită de către fundația Apache.

Astfel, în esență, canalul de comunicație bazat pe JMS este un broker de mesaje creat atunci când canalul de comunicație este pornit și căruia i se atașează o pereche IP/PORT.

Acest broker este responsabil pentru gestionarea unui topic şi managementul clienţilor. Ce este important de semnalat este faptul ca, în implementarea ActiveMQ nu este necesara crearea explicita a topicului la nivelul brokerului. Acesta va fi generat atunci când un client încearcă sa-l folosească pentru prima data.

Conectorii de ieşire, în cazul acestui canal de comunicaţie, sunt producători de mesaje. Ei se conectează la un topic predefinit şi trimit către acesta mesaje. Conectorii de intrare vor fi consumatori de mesaje conectata la acelaşi topic.

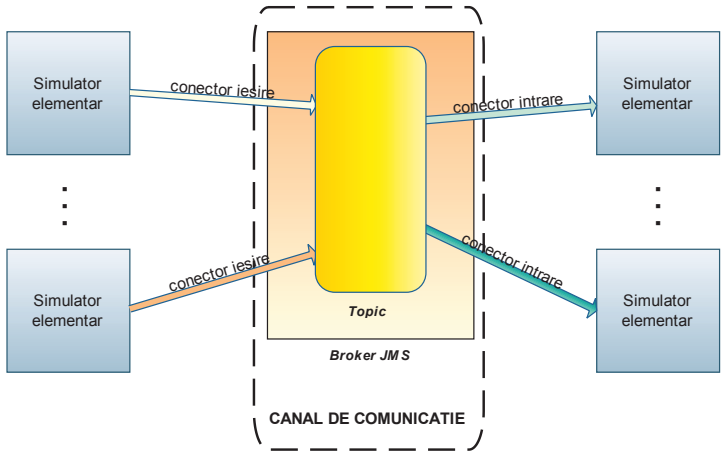

Figura 5.6 Canal de comunicaţie bazat pe JMS

5.3.3. Analiza eficientei canalelor de comunicaţie distribuite

În cazul canalelor de comunicaţie distribuite nu se mai poate folosi aceiaşi procedura de măsurare a timpului necesar pentru transportul datelor ca şi în cazul precedent deoarece componenta producător şi componenta consumator se afla pe maşini fizice diferite ce au ceas diferit. Deoarece timpii sunt măsuraţi la nivel de nanosecunda rezulta ca o diferenţă de o milisecunda între ceasurile celor doua maşini ar conduce la o eroare destul de mare. Pentru a putea totuşi verifica eficienta canalului de comunicaţie am combinat rezultatele din doua măsurători succesive, aşa cum se observă în diagrama din Figura 5.7.

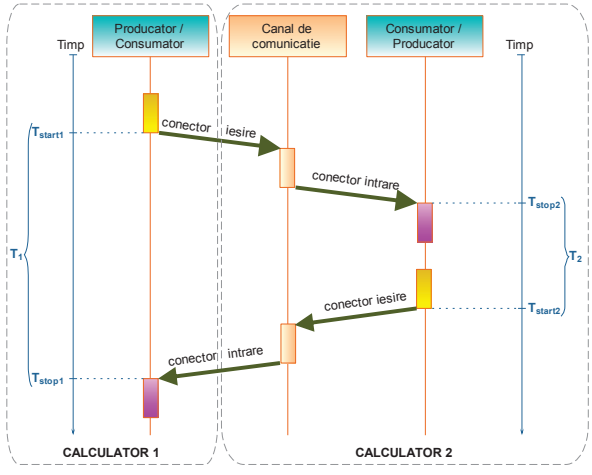

Figura 5.7 Măsurare timp de propagare canal de comunicaţie distribuit

Valorile de timp minime, maxime și medii sunt prezentate în Tabelul 5.3.

Canal de comunicație	**Dimensiune date (octeți)**	**Timp mediu (ns)**	**Timp maxim (ns)**	**Timp minim (ns)**	**Mai mic decât dublul valorii medii (%)**
TCP/IP	1	100150	319357	62994	99,53%
	2	98993	414708	55294	99,60%
	4	99323	402459	57744	99,66%
	8	98678	515848	56344	99,78%
	16	99974	362564	61944	99,86%
	32	100056	573943	59144	99,31%
ActiveMQ	1	208481	1144734	96940	95,13%
	2	219863	1195129	131236	95,87%
	4	217737	1163983	99390	96,39%
	8	216352	1142635	99040	96,86%
	16	228437	976051	124587	95,69%
	32	216820	1166432	97640	96,62%

Tabelul 5.3 Statistici canale de comunicație distribuite (I)

În Tabelul 5.3. sunt prezentate comparativ valorile medii măsurate pentru primele 1500 de mesaje și cele măsurate pentru următoarele 8500 de mesaje.

Se observă că în cazul acestor doua canale de comunicație timpul necesar pentru transferul de date este, în peste 99% din cazuri, cel mult 196µs pentru primul canal de comunicație și respectiv 400µs pentru cel de-al doilea canal de comunicație. Spre deosebire de canalele de comunicație locale se poate observa în aceste cazuri mai degrabă o stabilizare a timpului necesar pentru transfer în loc de o reducere consistenta cum a fost în cazul canalului de comunicație bazat pe cozi de mesaje.

Canal de comunicație	**Dimensiune date (octeți)**	**Timp mediu primele 1500 de mesaje (ns)**	**Timp mediu următoarele 8500 de mesaje (ns)**	**Diferența (%)**	**Mai mic decât dublul valorii medii pentru următoarele 8500 de mesaje (%)**
TCP/IP	1	108719	98637	9,27%	99,85%
	2	108858	97252	10,66%	99,58%
	4	109203	97579	10,64%	99,64%
	8	111327	96445	13,37%	99,74%
	16	110406	98133	11,12%	99,88%
	32	110150	98275	10,78%	99,48%
ActiveMQ	1	360825	181596	49,67%	99,28%
	2	362738	194650	46,34%	99,41%
	4	357268	193114	45,95%	99,09%
	8	354943	191895	45,94%	99,56%
	16	366116	204141	44,24%	97,86%
	32	355723	192308	45,94%	99,53%

Tabelul 5.4 Statistici canale de comunicație distribuite (II)

6. Studiu de caz – Simularea unei Ecluze Navigabile

6.1. **Prezentare generală**

O ecluză este o construcţie hidrotehnică amenajată pe traseul unei căi navigabile (canal artificial sau natural) sau la intrarea unui port cu maree, care asigură trecerea navelor între două suprafeţe de apă cu niveluri diferite.

În prezenta lucrare se urmăreşte implementarea unui simulator pentru o ecluza simpla. Primul pas în cadrul dezvoltării este reprezentat de analizarea instalaţiei ce se doreşte a fi simulata. Astfel rezultă următoarele componente pentru care trebuie realizate simulatoare elementare:

- Poarta plana de serviciu (PPS) – situata în amonte de ecluza;
- Poarta buscată de serviciu (PBS) – situata în aval de ecluza;
- Sasul(SAS) – camera în care staţionează navele pe parcursul ecluzării;
- Vane de umplere (VU1 şi VU2) – în număr de doua, situate în amonte de ecluza, sunt folosite pentru umplerea cu apa a sasului.
- Vane de golire (VG1 şi VG2) – în număr de doua, situate în aval de ecluza, sunt folosite pentru a golirea de apa a sasului.
- Nivel amonte (NAM) – nivelul apei în amonte de ecluză
- Nivel aval (NAV) – nivelul apei în aval de ecluză
- Control şi Comandă (CC) – componenta care controlează starea întregului sistem şi care determină activarea sau inactivarea porţilor şi a vanelor.

Se folosesc şi două componente oferite de sistem, şi anume generatorul semnal de ceas şi interfaţa grafică

În Tabelul 6.1 sunt prezentate componentele funcţionale ale sistemului de simulare pentru ecluza navigabila împreună cu variabilele de intrare şi de ieşire. În Tabelul 6.2 este prezentată componenta interfaţă grafică a simulatorului pentru ecluza navigabilă precum şi variabilele sale de intrare şi de ieşire.

Componenta	Variabile de intrare	Variabile de ieşire
Nivel amonte	q: cantitatea de apa pierduta	p_{amonte}: presiune amonte h_{amonte}: înălţimea coloanei de apa
Poarta plana (PP)	cmd_{pp}: comanda PP t: intervalul de timp	s_{pp}: starea PP poz_{ms}: poziţie motor stânga poz_{md}: poziţie motor dreapta
Vana de umplere 1 (VU1)	cmd_{vu1}: comanda VU1 t: intervalul de timp p_{sas}: presiune în sas p_{amonte}: presiune amonte	s_{vu1}: starea VU1 q: cantitatea de apa poz_{vu1}: poziţia VU1
Vana de umplere 2 (VU2)	cmd_{vu2}: comanda VU2 t: intervalul de timp p_{sas}: presiune în sas p_{amonte}: presiune amonte	s_{vu2}: starea VU2 q: cantitatea de apa poz_{vu2}: poziţia VU2
Sas	q: cantitatea de apa	P_{sas}: presiune în sas h_{sas}: înălţimea coloanei de apa

Vana de golire 1 (VG1)	cmd_{vg1}: comanda VG1 t: intervalul de timp p_{aval}: presiune aval p_{sas}: presiune în sas	s_{vg1}: starea VG1 q: cantitatea de apa poz_{vg1}: poziția VG1
Vana de golire 2(VG2)	cmd_{vg2}: comanda VG2 t: intervalul de timp p_{aval}: presiune aval p_{sas}: presiune în sas	s_{vg2}: starea VG2 q: cantitatea de apa poz_{vg2}: poziția VG2
Poarta buscată (PB)	cmd_{pb}: comanda PB t: intervalul de timp	s_{pb}: starea PB poz_{ps}: poziție poarta stânga poz_{pd}: poziție poarta dreapta
Nivel aval	q: cantitatea de apa primita	p_{aval}: presiune amonte h_{aval}: înălțimea coloanei de apa
Generator semnal de ceas	S: comanda start ceas	t: interval de timp
Control și Comanda	h_{amonte}: înălțimea coloanei de apa s_{pp}: starea PP s_{vu1}: starea VU1 s_{vu2}: starea VU2 h_{sas}: înălțimea coloanei de apa s_{vg1}: starea VG1 s_{vg2}: starea VG2 s_{pb}: starea PB h_{aval}: înălțimea coloanei de apa	cmd_{pp}: comanda PP cmd_{vu1}: comanda VU1 cmd_{vu2}: comanda VU2 cmd_{vg1}: comanda VG1 cmd_{vg2}: comanda VG2 cmd_{pb}: comanda PB

Tabelul 6.1 Componentele funcționale ale simulatorului pentru ecluza navigabilă

Componenta	**Variabile de intrare**	**Variabile de ieșire**
Interfața grafica	h_{amonte}: înălțimea coloanei de apa s_{pp}: starea PP poz_{ms}: poziție motor stânga poz_{md}: poziție motor dreapta s_{vu1}: starea VU1 poz_{vu1}: poziția VU1 s_{vu2}: starea VU2 poz_{vu2}: poziția VU2 h_{sas}: înălțimea coloanei de apa s_{vg1}: starea VG1 poz_{vg1}: poziția VG1 s_{vg2}: starea VG2 poz_{vg2}: poziția VG2 s_{pb}: starea PB poz_{ps}: poziție poarta stânga poz_{pd}: poziție poarta dreapta h_{aval}: înălțimea coloanei de apa	S: comanda start ceas cmd_{pp}: comanda PP cmd_{vu1}: comanda VU1 cmd_{vu2}: comanda VU2 cmd_{vg1}: comanda VG1 cmd_{vg2}: comanda VG2 cmd_{pb}: comanda PB

Tabelul 6.2 Interfața grafică a simulatorului pentru ecluza navigabilă

Pornind de la aceste tabele se poate realiza o reprezentare grafica a componentelor simulatorului așa cum se observa în Figura 6.1.

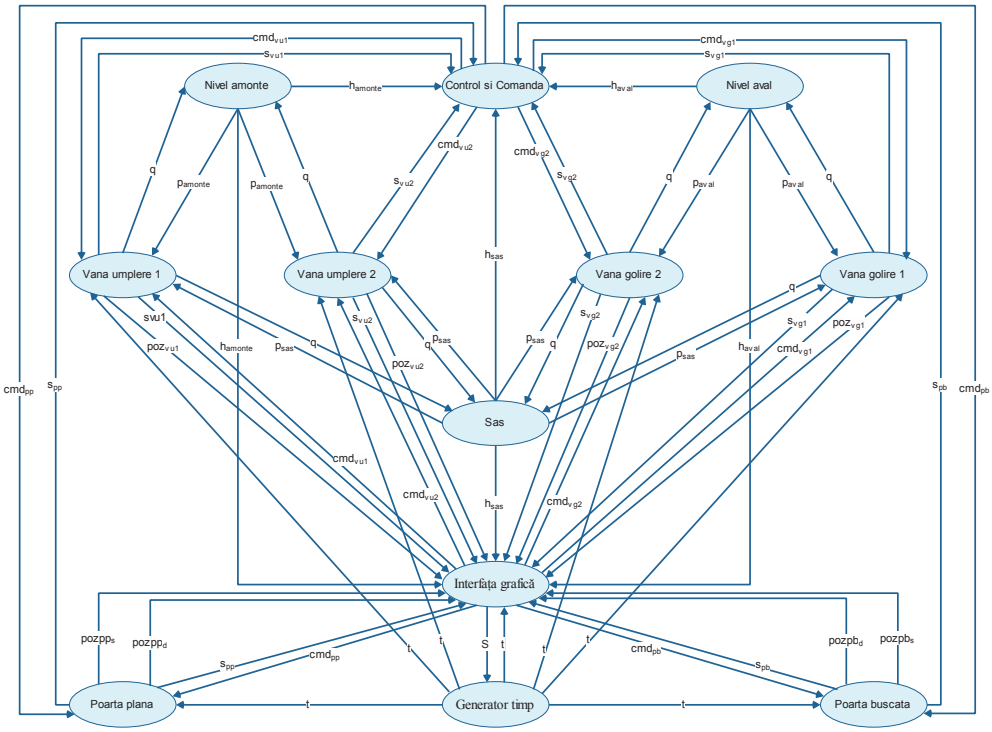

Figura 6.1 Diagrama componente ecluza

Numărul maxim de canale de comunicație necesare pentru realizarea simulatorului, așa cum se poate deduce din Figura 6.1, este de 62, un număr foarte mare, ce ar conduce la o viteza de execuție redusa deoarece foarte mult timp procesor se va folosi pentru buna execuție a canalelor de comunicație. Prin aplicarea algoritmului de minimizare a numărului canalelor de comunicație se obține numărul optim necesar egal cu 15. Pe baza acestor informații se poate genera fișierul sau, după caz, fișierele de configurație ce vor fi folosite pentru realizarea simulatorului.

6.2. **Simulator Nivel**

Aceasta componenta este folosita în cadrul sistemului pentru a simula comportamentul coloanei de apa aflata în amonte de ecluza și, respectiv, în aval de ecluza. Schema bloc a acestui simulator este prezentata în Figura 6.2.

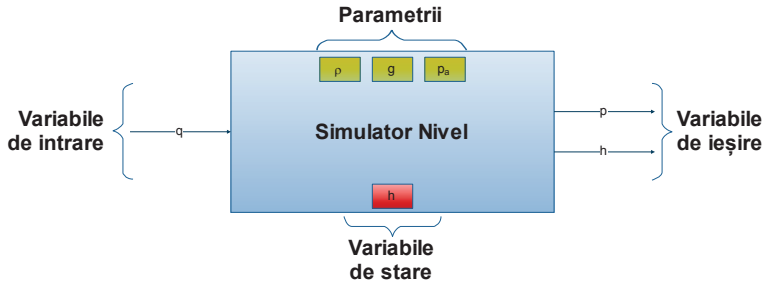

Figura 6.2 Simulator Nivel - schema bloc

În Tabelul 6.3 sunt prezentați parametrii și variabilele folosite de către simulatorul nivel.

Parametrii	p_a: presiunea atmosferica, ρ: densitatea apei, g: accelerația gravitațională,
Variabile de intrare	q: cantitatea de apa primita sau pierduta
Variabile de stare	h: înălțimea coloanei de apa
Variabile de ieșire	p: presiunea în punctul cel mai de jos h: înălțimea coloanei de apa

Tabelul 6.3 Parametrii și variabile pentru Simulatorul Nivel

Algoritmul de funcționare al acestei componente este prezentat în continuare:

```
Pentru orice valoare a variabilei de intrare „q”
      Emite variabila de ieşire „p”
      Emite variabila de ieşire „h”
```

Algoritmul prezentat mai sus este implementat în Java sub forma unei clase ce extinde simulatorul elementar abstract. În cadrul simulatorului pentru ecluza se vor folosi doua instanțe ale acestei componente, una pentru nivelul amonte, și una pentru nivelul aval.

6.3. **Simulator Poarta Plana**

Aceasta componenta este folosita în cadrul sistemului pentru a simula comportamentul porții aflate în amonte și care se numește „Poarta Plana de Serviciu”. Acest echipament este format dintr-o poarta rigida de metal ce se poate deplasa vertical cu ajutorul a doua motoare situate la capetele din stânga și respectiv din dreapta. Cele doua motoare sunt independente unul de altul dar sunt sincronizate cu ajutorul unui dispozitiv de automatizare a procesului.

Schema bloc a acestui simulator este prezentata în Figura 6.3

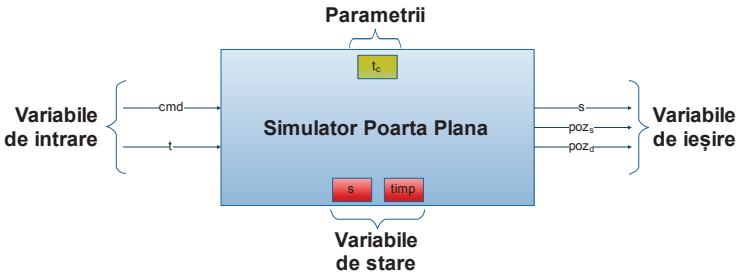

Figura 6.3 Simulator Poarta Plana - schema bloc

În Tabelul 6.4 sunt prezentați parametrii și variabilele folosite de către simulatorul poarta plana.

Parametrii	t_c: factorul de compresie al timpului, folosit pentru a mari viteza de închidere sau de deschidere a porții
Variabile de intrare	cmd: comanda recepționată de la operator sau de la *„ControlComandă”* t: intervalul de timp
Variabile de stare	s: starea simulatorului timp: timpul scurs de la începerea manevrei
Variabile de ieșire	s: starea simulatorului poz_s: poziția motorului din stânga în intervalul [0,1] poz_d: poziția motorului din dreapta în intervalul [0,1]

Tabelul 6.4 Parametrii și variabile pentru Simulatorul Poarta Plana

În esență acest simulator este o mașina de stări controlata cu ajutorul variabilelor de intrare. Tranziția între stări se face în funcție de variabilele de intrare. Din punct de vedere al implementării acest simulator prezintă doua fluxuri de date, unul controlat de variabila de intrare „*cmd*" iar celalalt de variabila „*t*". Primul flux are efecte numai asupra stării componentei, în timp ce cel de-al doilea este responsabil cu calcularea noilor valori pentru variabilele de ieșire și trimiterea lor prin intermediul canalelor de comunicație. Evoluția stărilor mașinii în funcție de comanda primita este determinata conform algoritmului descris în continuare:

```
daca cmd este „Activare simulator" atunci
  stare ← „Activat"
altfel daca cmd este „Dezactivare simulator" atunci
  stare ← „Dezactivat"
altfel daca cmd este „Pornire execuție" atunci
| daca stare este „Deschis" sau „Oprit în deschidere" atunci
|   stare ← „În închidere"
| altfel daca stare este „Închis" sau „Oprit în închidere" atunci
|   stare ← „În deschidere"
| altfel
|   stare nu se modifica
└#
altfel daca cmd este „Oprire execuție" atunci
| daca stare este „În Deschidere" atunci
|   stare ← „Oprit în deschidere"
| altfel daca stare este „În Închidere" atunci
|   stare ← „Oprit în închidere"
| altfel
|   stare nu se modifica
└#
```

Cel de-al doilea flux de date, controlat de variabila *„t"*, este responsabil, în principal, de generarea valorilor variabilelor de ieșire „poz_s" și „poz_d" corespunzătoare poziției relative a motoarelor stânga și dreapta, în intervalul [0,1], dar și de actualizarea stării componentei.

În Tabelul 6.5 sunt sintetizate funcțiile de calcul a poziției relative raportata la timp pentru operația de deschidere a porții plane iar în Tabelul 6.6 sunt sintetizate funcțiile de calcul a poziției relative raportata la timp pentru operația de închidere a porții plane.

	Interval de timp (s)	**Poziție relativa la început**	**Poziție relativa la final**	**Funcție de calcul a poziției relative**
Motor Dreapta	0-39	0.997	1.0	$\text{poz}_\text{d} = \frac{0.003}{39} * \text{t} + 0.997$
	39-365	1.0	0.00877561	$\text{poz}_\text{d} = -\frac{0.99122439}{326} * \text{t} + 1.118582059$
	365-372	0.00877561	0.0	$\text{poz}_\text{d} = -\frac{0.00877561}{7} * \text{t} + 0.466360982$
Motor Stânga	0-39	0.997	1.0	$\text{poz}_\text{s} = \frac{0.003}{39} * \text{t} + 0.997$
	39-365	1.0	0.008689768	$\text{poz}_\text{s} = -\frac{0.99131}{326} * \text{t} + 1.118592328$
	365-372	0.008689768	0.0	$\text{poz}_\text{s} = -\frac{0.00869}{7} * \text{t} + 0.461799115$

Tabelul 6.5 Funcțiile de calcul a poziției motoarelor porții plane în timpul operației de deschidere

	Interval de timp (s)	Poziție relativa la început	Poziție relativa la final	Funcție de calcul a poziției relative
Motor Dreapta	0-107	0.0	0.049234796	$\text{poz}_\text{d} = \frac{0.049234796}{107} * \text{t}$
	107-507	0.049234796	0.970221273	$\text{poz}_\text{d} = \frac{0.920986478}{400} * \text{t} - 0.197129087$
	507-571	0.970221273	1.0	$\text{poz}_\text{d} = \frac{0.029778727}{64} * \text{t} + 0.734317921$
	571-637	1.0	0.997	$\text{poz}_\text{d} = -\frac{0.0029083}{66} * \text{t} + 1.025161202$
Motor Stânga	0-107	0.0	0.050291793	$\text{poz}_\text{s} = \frac{0.050291793}{107} * \text{t}$
	107-507	0.050291793	0.974861794	$\text{poz}_\text{s} = \frac{0.924570}{400} * \text{t} - 0.197030682$
	507-571	0.974861794	1.0	$\text{poz}_\text{s} = \frac{0.025138}{64} * \text{t} + 0.77572007$
	571-637	1.0	0.997	$\text{poz}_\text{s} = -\frac{0.002542}{66} * \text{t} + 1.021991931$

Tabelul 6.6 Funcțiile de calcul a poziției motoarelor porții plane în timpul operației de închidere

Algoritmul ce descrie funcționarea fluxului de date responsabil cu generarea poziției relative a porții plane este:

```
daca stare este „În deschidere” atunci
| timp ← timp + t * tc
| calculează pozs folosind formula corespunzătoare pentru valoarea lui timp
| calculează pozd folosind formula corespunzătoare pentru valoarea lui timp
| daca timp este mai mare sau egal decât 372s atunci
| | timp ← 0
| | pozs ← 0.0
| | pozd ← 0.0
| | stare ← „Deschis”
└# └#
altfel daca stare este „În închidere” atunci
| timp ← timp + t * tc
| calculează pozs folosind formula corespunzătoare pentru valoarea lui timp
| calculează pozd folosind formula corespunzătoare pentru valoarea lui timp
| daca timp este mai mare sau egal decât 637s atunci
| | timp ← 0
| | pozs ← 0.997
| | pozd ← 0.997
| | stare ← „Închis”
└# └#
altfel
   nu se modifica nimic
trimite pozs, pozd şi stare către conectorii de ieşire
```

Algoritmii prezentați mai sus sunt implementați în Java sub forma unei clase ce extinde simulatorul elementar abstract.

6.4. **Simulator Poarta Buscata**

Aceasta componenta este folosita în cadrul sistemului pentru a simula comportamentul porții aflate în aval și care se numește „Poarta Buscată de Serviciu”. Spre deosebire de poarta plana, poarta buscată de serviciu este formata din doua porți ce se deplasează independent, una spre stânga și cealaltă spre dreapta. Schema bloc a acestui simulator este prezentata în Figura 6.4.

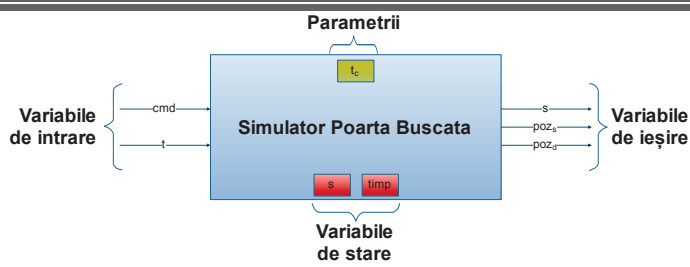

Figura 6.4 Simulator Poarta Buscata - schema bloc

Parametrii	t_c: factorul de compresie al timpului, folosit pentru a mari viteza de închidere sau de deschidere a porții
Variabile de intrare	cmd: comanda recepționată de la operator sau de la *„ControlComandă”* t: intervalul de timp
Variabile de stare	s: starea simulatorului timp: timpul scurs de la începerea manevrei
Variabile de ieșire	s: starea simulatorului poz_s: poziția porții din stânga în intervalul [0,1] poz_d: poziția porții din dreapta în intervalul [0,1]

Tabelul 6.7 Parametrii și variabile pentru Simulatorul Poarta Buscata

În Tabelul 6.7 sunt prezentați parametrii și variabilele folosite de către simulatorul poarta buscată.

Ca și în cazul simulatorului pentru poarta plana și acest simulator este o mașina de stări controlata cu ajutorul variabilelor de intrare. Din punct de vedere al implementării și acest simulator, la fel ca cel pentru poarta plana, prezintă doua fluxuri de date, unul controlat de variabila de intrare „*cmd*” iar celalalt de variabila „*t*”. Fluxul de date controlat de variabila de intrare „*cmd*” determina evoluția stării componentei, fără a o trimite prin intermediul canalelor de comunicație. Evoluția stărilor mașinii în funcție de comanda primita este determinata conform algoritmului descris în continuare, identic de altfel cu cel folosit pentru poarta plana:

```
daca cmd este „Activare simulator” atunci
  stare ← „Activat”
altfel daca cmd este „Dezactivare simulator” atunci
  stare ← „Dezactivat”
altfel daca cmd este „Pornire execuție” atunci
| daca stare este „Deschis” sau „Oprit în deschidere” atunci
|   stare ← „În închidere”
| altfel daca stare este „Închis” sau „Oprit în închidere” atunci
|   stare ← „În deschidere”
| altfel
|   stare nu se modifica
└#
altfel daca cmd este „Oprire execuție” atunci
| daca stare este „În Deschidere” atunci
|   stare ← „Oprit în deschidere”
| altfel daca stare este „În Închidere” atunci
|   stare ← „Oprit în închidere”
| altfel
|   stare nu se modifica
└#
```

Cel de-al doilea flux de date, controlat de variabila *„t”*, este responsabil, în principal, de generarea valorilor variabilelor de ieșire „poz_s” și „poz_d” corespunzătoare poziției relative a porților din stânga și respectiv dreapta, în intervalul [0,1], dar și de actualizarea stării componentei

	Interval de timp (s)	**Poziție relativa la început**	**Poziție relativa la final**	**Funcție de calcul a poziției relative**
Poarta Dreapta	0-26	1	0.982658772	$\text{poz}_\text{d} = -\frac{0.017341228}{26} * \text{t} + 1$
	26-83	0.982658772	0.92464949	$\text{poz}_\text{d} = -\frac{0.058009282}{57} * \text{t} + 1.009119147$
	83-317	0.92464949	0.021996388	$\text{poz}_\text{d} = -\frac{0.902653103}{234} * \text{t} + 1.244821317$
	317-336	0.021996388	0	$\text{poz}_\text{d} = -\frac{0.021996388}{19} * \text{t} + 0.388988749$
Poarta Stânga	0-16	1	0.992406324	$\text{poz}_\text{s} = -\frac{0.007594}{16} * \text{t} + 1$
	16-113	0.992406324	0.873866804	$\text{poz}_\text{s} = -\frac{0.118540}{97} * \text{t} + 1.011959235$
	113-306	0.873866804	0.004031384	$\text{poz}_\text{s} = -\frac{0.869835}{193} * \text{t} + 1.383148682$
	306-336	0.004031384	0	$\text{poz}_\text{s} = -\frac{0.004031}{30} * \text{t} + 0.0451515$

Tabelul 6.8 Funcțiile de calcul a poziției porților ce alcătuiesc poarta buscată, în timpul operației de deschidere

. Ca și în cazul anterior, dependenta de timp a poziției celor doua porți este determinata prin analizarea unor seturi de date extrase de la o ecluza. În Tabelul 6.8 sunt sintetizate funcțiile de calcul a poziției relative raportata la timp pentru operația de deschidere a porții buscate iar în Tabelul 6.9 sunt sintetizate funcțiile de calcul a poziției relative raportata la timp pentru operația de închidere a porții buscate.

	Interval de timp (s)	**Poziție relativa la început**	**Poziție relativa la final**	**Funcție de calcul a poziției relative**
Poarta Dreapta	0-40	0	0.03983817	$\text{poz}_\text{d} = \frac{0.03983817}{40} * \text{t}$
	40-310	0.03983817	0.934228068	$\text{poz}_\text{d} = \frac{0.894389898}{270} * \text{t} - 0.092664037$
	310-408	0.934228068	1	$\text{poz}_\text{d} = \frac{0.065771932}{98} * \text{t} + 0.726173999$
Poarta Stânga	0-30	0	0.029139296	$\text{poz}_\text{s} = \frac{0.029139}{30} * \text{t}$
	30-286	0.029139296	0.892084334	$\text{poz}_\text{s} = \frac{0.862945}{256} * \text{t} - 0.071987075$
	286-408	0.892084334	1	$\text{poz}_\text{s} = \frac{0.107916}{122} * \text{t} + 0.639101707$

Tabelul 6.9 Funcțiile de calcul a poziției porților ce alcătuiesc poarta buscată, în timpul operației de închidere

Algoritmul ce descrie funcționarea fluxului de date responsabil cu generarea poziției relative a porții buscate este:

```
daca stare este „În deschidere” atunci
| timp ← timp + t * tc
| calculează pozs folosind formula corespunzătoare pentru valoarea lui timp
| calculează pozd folosind formula corespunzătoare pentru valoarea lui timp
| daca timp este mai mare sau egal decât 336s atunci
| | timp ← 0
| | pozs ← 0.0
| | pozd ← 0.0
| | stare ← „Deschis”
└# └#
altfel daca stare este „În închidere” atunci
| timp ← timp + t * tc
| calculează pozs folosind formula corespunzătoare pentru valoarea lui timp
| calculează pozd folosind formula corespunzătoare pentru valoarea lui timp
| daca timp este mai mare sau egal decât 408s atunci
| | timp ← 0
| | pozs ← 1.0
| | pozd ← 1.0
| | stare ← „Închis”
└# └#
altfel
   nu se modifica nimic
trimite pozs, pozd şi stare către conectorii de ieşire
```

Algoritmii prezentați mai sus sunt implementați în Java sub forma unei clase ce extinde simulatorul elementar abstract.

6.5. **Simulator Vana**

Aceasta componenta este folosita în cadrul sistemului pentru a simula comportamentul unei vane folosita pentru umplerea sau respectiv golirea sasului. O vana este considerata a fi o conducta ce poate fi obturata total sau parțial controlându-se astfel cantitatea de apa ce o străbate. Cantitatea de apă ce străbate o vană va fi calculată folosind dimensiunile acesteia precum și diferența de presiune la cele două capete.Schema bloc a acestui simulator este prezentata în Figura 6.5.

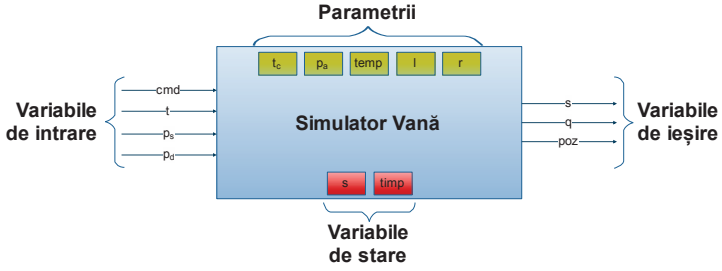

Figura 6.5 Simulator Vană - schema bloc

În Tabelul 6.10 sunt prezentați parametrii și variabilele folosite de către simulatorul vană.

Ca și în cazul simulatorului pentru poarta plana și acest simulator este o mașina de stări controlata cu ajutorul variabilelor de intrare. Tranziția între stări se face numai în funcție de variabilele de intrare „t” și „cmd” care controlează fiecare cate un flux de date.

Parametrii	t_c: factorul de compresie al timpului, folosit pentru a mari viteza de închidere sau de deschidere a vanei p_a: presiunea atmosferică folosită pentru a determina daca exista apa la capetele vanei

	temp: temperatura apei folosita pentru a determina vâscozitatea l: lungimea vanei r: raza secțiunii vanei
Variabile de intrare	cmd: comanda recepționată de la operator sau de la *„ControlComandă”* t: intervalul de timp p_s: presiunea la capătul din stânga al vanei p_d presiunea la capătul din dreapta al vanei
Variabile de stare	s: starea simulatorului timp: timpul scurs de la începerea manevrei
Variabile de ieșire	s: starea simulatorului q: cantitatea de apa care a străbătut vana de la stânga la dreapta în intervalul de timp „t” poz: poziția curentă a vanei în intervalul [0,1]

Tabelul 6.10 Parametrii și variabile pentru Simulatorul Vană

Fluxul de date controlat de variabila de intrare „*cmd*” determina evoluția stării componentei, fără a o trimite prin intermediul canalelor de comunicație. Evoluția stărilor mașinii în funcție de comanda primita este determinata conform algoritmului descris în continuare, identic de altfel cu cel folosit pentru poarta plana și poarta buscată:

```
daca cmd este „Activare simulator” atunci
  stare ← „Activat”
altfel daca cmd este „Dezactivare simulator” atunci
  stare ← „Dezactivat”
altfel daca cmd este „Pornire execuție” atunci
| daca stare este „Deschis” sau „Oprit în deschidere” atunci
|   stare ← „În închidere”
| altfel daca stare este „Închis” sau „Oprit în închidere” atunci
|   stare ← „În deschidere”
| altfel
|   stare nu se modifica
└#
altfel daca cmd este „Oprire execuție” atunci
| daca stare este „În Deschidere” atunci
|   stare ← „Oprit în deschidere”
| altfel daca stare este „În Închidere” atunci
|   stare ← „Oprit în închidere”
| altfel
|   stare nu se modifica
└#
```

Cel de-al doilea flux de date, controlat de variabila *„t”*, este responsabil, în principal, de generarea valorilor variabilelor de ieșire „poz” și „q” corespunzătoare poziției relative a vanei, în intervalul [0,1] și respectiv cantității de apa ce străbate vana dintr-o parte în alta și de actualizarea stării componentei.

	Interval de timp (s)	**Poziție relativa la început**	**Poziție relativa la final**	**Funcție de calcul a poziției relative**
Vana	0-6	0	0.005753544	$\text{poz} = \frac{0.005754}{6} * \text{t}$
	6-225	0.005753544	0.990842999	$\text{poz} = \frac{0.985089}{219} * \text{t} - 0.021235208$
	225-230	0.990842999	1	$\text{poz} = \frac{0.009157}{5} * \text{t} + 0.578777976$

Tabelul 6.11 Funcțiile de calcul a poziției vanei, în timpul operației de deschidere

În Tabelul 6.11 sunt sintetizate funcțiile de calcul a poziției relative raportata la timp pentru operația de deschidere a vanei iar în Tabelul 6.12 sunt sintetizate funcțiile de calcul a poziției relative raportata la timp pentru operația de închidere a vanei.

	Interval de timp (s)	Poziție relativa la început	Poziție relativa la final	Funcție de calcul a poziției relative
Vana	0-5	1	0.994925505	$\text{poz} = -\frac{0.005074}{5} * \text{t} + 1$
	5-189	0.994925505	0.025606103	$\text{poz} = -\frac{0.969319}{184} * \text{t} + 1.021265706$
	189-214	0.025606103	0	$\text{poz} = -\frac{0.025606}{25} * \text{t} + 0.219188241$

Tabelul 6.12 Funcțiile de calcul a poziției vanei, în timpul operației de închidere

Algoritmul rezultat este prezentat în continuare.

```
daca stare este „În deschidere” atunci
| timp ← timp + t * tc
| calculează poz folosind formula corespunzătoare pentru valoarea lui timp
| daca timp este mai mare sau egal decât 230s atunci
| | timp ← 0
| | poz ← 1.0
| | stare ← „Deschis”
└#└#
altfel daca stare este „În închidere” atunci
| timp ← timp + t * tc
| calculează poz folosind formula corespunzătoare pentru valoarea lui timp
| daca timp este mai mare sau egal decât 214s atunci
| | timp ← 0
| | poz ← 0.0
| | stare ← „Închis”
└#└#
altfel
   nu se modifica nimic
```

Cea de-a doua secțiune a fluxului de date controlat de variabila „t” se refera la calcularea cantității de apa ce străbate vana dintr-o parte în alta în intervalul de timp dat. Aceasta valoare nu depinde de starea vanei, ci doar de dimensiuni, presiunea la capete, temperatura și poziția ei. Practic vana este considerata o țeava pentru care exista posibilitatea de obturare. De aceea formula folosita pentru calcularea cantității de apa ce străbate vana într-un interval de timp este data de legea Hagen-Poiseuille. Algoritmul utilizat este prezentat în continuare:

```
calculează poz și stare folosind algoritmul de mai sus
calculează q folosind legea Hagen-Poiseuille
trimite poz, stare și q către conectorii de ieșire.
```

În cadrul simulatorului pentru ecluza se vor folosi patru instanțe ale acestei componente, doua pentru vanele de umplere, ce fac legătura între nivelul amonte și sas-ul ecluzei, și doua pentru vanele de golire, ce fac legătura între sasul ecluzei și nivelul aval.

6.6. **Simulator Sas**

Sasul ecluzei este locul în care staționează navele pe durata operației de ecluzare. Sasul este de fapt un bazin ce conține o coloana de apa a cărei înălțime variază între valoarea nivelului din amonte și valoarea nivelului din aval. Schema bloc este prezentata în Figura 6.6.

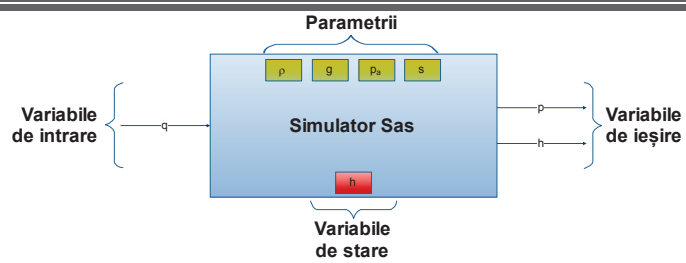

Figura 6.6 Simulator Sas - schema bloc

În Tabelul 6.13 sunt prezentați parametrii și variabilele folosite de către simulatorul sas.

Parametrii	p_a: presiunea atmosferica, ρ: densitatea apei, g: accelerația gravitațională, s: suprafața sasului
Variabile de intrare	q: cantitatea de apa primita sau pierduta
Variabile de stare	h: înălțimea coloanei de apa
Variabile de ieșire	p: presiunea în punctul cel mai de jos h: înălțimea coloanei de apa

Tabelul 6.13 Parametrii și variabile pentru Simulatorul Sas

Algoritmul de funcționare al acestei componente este prezentat în continuare

```
calculează noua valoare a lui „h”
calculează noua valoare a lui „p”
emite variabila de ieşire „p”
emite variabila de ieşire „h”
```

6.7. **Control și Comanda**

Componenta control și comanda este folosita pentru a activa sau dezactiva simulatoare ce accepta comenzi de la utilizator, Sunt vizate simulatorul Poarta Plana, Simulatorul Poarta Buscata și Simulatorul Vana. Deși aceasta componenta nu este un simulator, ea este construita pe același principiu. Schema bloc a acestei componente este prezentata în Figura 6.7.

Figura 6.7 Control și Comanda - schema bloc

În Tabelul 6.14 sunt prezentați parametrii și variabilele folosite de către componenta control și comanda.

Parametrii	-
Variabile de intrare	$stare_{vu1}$: starea vanei de umplere 1 $stare_{vu2}$: starea vanei de umplere 2 $stare_{vg1}$: starea vanei de golire 1 $stare_{vg2}$: starea vanei de golire 2 $stare_{pp}$: starea porții plane $stare_{pb}$: starea porții buscate

	$nivel_{amonte}$: nivelul apei în amonte de ecluza $nivel_{aval}$: nivelul apei în aval de ecluza $nivel_{sas}$: nivelul apei în sas
Variabile de stare	-
Variabile de ieșire	cmd_{vu}: comanda trimisa vanelor de umplere cmd_{vg}: comanda trimisa vanelor de golire cmd_{pp}: comanda trimisa porții plane cmd_{pb}: comanda trimisa porții buscate

Tabelul 6.14 Parametrii și variabile pentru Control Comanda

Aceasta componenta primește starea tuturor vanelor și porților dar și nivelul amonte, aval și din sas și determina pe baza acestor informații ce componentă poate accepta sau nu comenzi de la utilizator. Ea are o funcționare automata, neinfluențata de utilizator. Algoritmul folosit este:

```
daca stare_vg1, stare_vg2, stare_pp și stare_pb sunt egale cu „Închis” atunci
  cmd_vu ← „Activat”
altfel
  cmd_vu ← „Dezactivat”
daca stare_vu1, stare_vu2, stare_pp și stare_pb sunt egale cu „Închis” atunci
  cmd_vg ← „Activat”
altfel
  cmd_vg ← „Dezactivat”
daca stare_vq1, stare_vq2 și stare_pb sunt egale cu „Închis” și
     stare_vu1 și stare_vu2 sunt egale cu „Deschis” și
     |nivel_amonte - nivel_sas| ≈ 0 atunci
  cmd_pp ← „Activat”
altfel
  cmd_pp ← „Dezactivat”
daca stare_vu1, stare_vu2 și stare_pp sunt egale cu „Închis” și
     stare_vq1 și stare_vq2 sunt egale cu „Deschis” și
     |nivel_aval - nivel_sas| ≈ 0 atunci
  cmd_pb ← „Activat”
altfel
  cmd_pb ← „Dezactivat”
trimite valorile variabilelor cmd_vu, cmd_vg, cmd_pp, cmd_pb
```

6.8. **Interfața Grafică**

Componenta interfața grafica este folosita pentru a oferi posibilitatea utilizatorului sa interacționeze cu simulatorul. Interfața grafica este realizata folosind componenta oferita de sistem și descrisa în capitolul 4.4.1. Pentru fiecare simulator (cu excepția componentei control și comanda) s-au realizat componente grafice corespondente. S-au realizat pentru unele simulatoare cate doua componente grafice cu semnificația „vedere aeriana” și „vedere laterala” prezentate în Figura 6.8, și Figura 6.9.

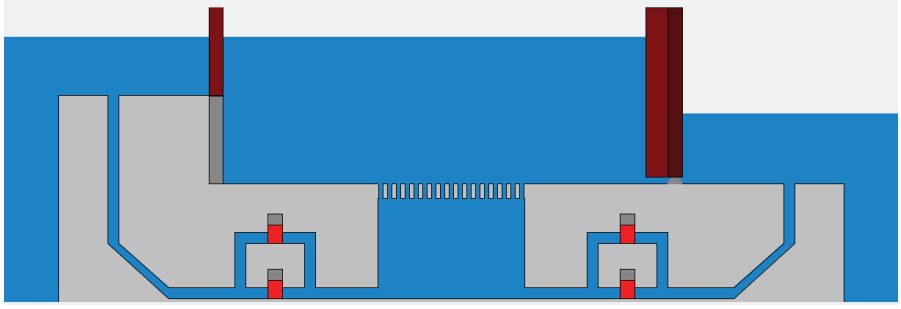

Figura 6.8 Interfața grafica – vedere laterala

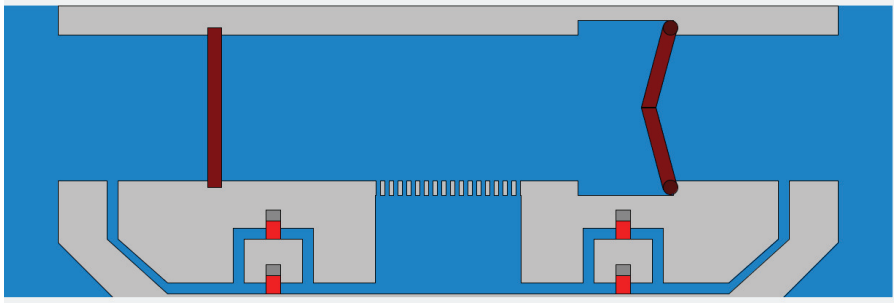

Figura 6.9 Interfața grafica – vedere aeriană

Deși aceasta interfața grafica pare a fi realizata monolitic ea este de fapt formata din diverse componente, majoritatea având legătură directa cu simulatoarele elementare. Exista însă și elemente cu rol pur grafic cum ar fi legăturile între vane și sas și respectiv vane și nivelul amonte sau aval.

7. Concluzii

Sistemul propus poate reprezenta o soluţie fiabila pentru realizarea de simulatoare pentru diferite instalaţii industriale şi nu numai. Astfel, acest sistem poate fi folosit și pentru implementarea de simulatoare pentru scheme logice, reţele de calculatoare etc. ce pot fi folosite în mediul academic ca material didactic acolo unde folosirea unor echipamente reale este imposibila.

Folosind sistemul bazat pe canale de comunicație și conectori și datorita gradului mare de configurare, un utilizator poate să folosească puterea de calcul a unui număr mare de calculatoare pentru a realiza un simulator eficient. Un număr de patru canale de comunicație sunt oferite în acest moment, dar acest lucru nu limitează dezvoltarea de alte elemente, integrarea acestora efectuându-se foarte uşor. Canalele de comunicație oferite în prezent permit atât interconectarea entităţilor aflate pe aceiaşi maşină cat și a celor aflate pe maşini diferite iar timpul necesar pentru propagarea mesajelor este, în general, de maxim 21 de microsecunde pentru canalele locale de comunicație și respectiv între 200 și 400 de microsecunde pentru celelalte, în funcție de implementarea aleasă.

Folosirea unor tehnologii standardizate (cum este JMS) în cazul canalelor de comunicație permite realizarea de conectori în alte limbaje de programare decât Java fapt ce poate avea un impact important atât asupra vitezei de execuție cat și oferă posibilitatea de a integra relativ uşor componente reale în interiorul simulatorului global.

În plus, sistemul oferă și posibilitatea de a folosi o platforma pentru dezvoltarea simulatoarelor elementare, întârzierea introdusa fiind de maxim 34 de microsecunde, dar având avantajul ca se pot realiza componentele foarte rapid și fără a fi nevoie de cunoştinţe detaliate ale întregului sistem.

Un alt avantaj al acestei abordări o reprezintă și faptul ca în cazul simulatoarelor elementare nu există, prin design, nici o legătură între partea funcţională și interfața grafica. Astfel, aceste simulatoare pot rula fără probleme chiar și fără o interfaţă grafica sau, mai mult, aceasta poate fi simulata cu ajutorul unor componente dezvoltate special pentru aceasta, astfel încât sa genereze comenzi conform unui scenariu prestabilit pentru a putea testa comportamente sau chiar componente reale.

Folosirea Java s-a dovedit a fi pana la urma un avantaj deoarece ultimele evoluţii ale acestei platforme au oferit un plus de stabilitate și de viteza pe lângă uşurinţa de programare care era specifica acestui limbaj. Mai mult, independenta de platforma a Java precum și utilitarele open-source existente oferă costuri de dezvoltare mult reduse fata de alte limbaje de programare sau platforme existente în acest moment pe piaţă.

Simulatorul pentru ecluza implementat folosind arhitectura studiata este un exemplu al utilității acesteia. Folosirea simulatorului elementar abstract pentru toate operațiile comune a permis realizarea de componente axate strict funcționalitatea dorita fapt ce a redus considerabil timpul de dezvoltare. Arhitectura sistemului îi permite unui administrator sa verifice diferite distribuiri spațiale pentru a putea alege topologia optima folosind astfel la maxim toate resursele hardware disponibile.

Datorită modularității sale sistemul permite și interfațarea cu echipamente reale prin simpla înlocuire a unui simulator elementar cu un modul software ce poate comunica cu sistemul hardware respectiv. Astfel un operator uman poate fi antrenat folosind un simulator iar atunci când este pregătit va putea folosii echipamentele reale păstrând aceiași interfață grafică cu care este deja obișnuit reducându-se astfel riscul de a greși și timpul necesar pentru familiarizarea cu instalația pe care trebuie să o conducă sau supravegheze.

7.1. **Contribuții personale**

Contribuțiile personale sunt prezentate pe parcursul prezentei teze, după cum urmează:

În capitolul 3:

- Proiectarea generală a sistemului de simulare și identificarea funcțiilor generale ale celor doua tipuri de module software – simulatoarele elementare și canalele de comunicație (3.1)
- Dezvoltarea unui algoritm necesar pentru reducerea numărului canalelor de comunicație pentru a eficientiza cantitatea de memorie utilizată și timpul de execuție al simulării (3.2)
- Proiectarea generala a unui modul software ce poate fi integrat în sistem și ce poate fi folosit pentru comunicația cu echipamentele fizice (3.3.1)
- Proiectarea generala a unui modul software ce poate fi integrat în sistem și ce poate fi folosit pentru interfața cu operatorii umani (3.3.2)
- Proiectarea și implementarea unui modul software folosit pentru citirea și interpretarea fișierului de configurație al simulatorului (3.4.1)
- Proiectarea și implementarea unui simulator pentru o instalație industriala simpla, formata din doua bazine ce conțin cantități diferite de apa și care sunt interconectate între ele prin intermediul unui robinet

În capitolul 4:

- Analizarea simulatoarelor elementare (SE), văzute ca sisteme dinamice, pentru identificarea funcționalităților comune (4.1)
- Proiectarea și implementarea unei componente software, numita Simulatorul Elementar Abstract (SEA), ce însumează funcționalitățile comune ale tuturor SE folosite pentru realizarea sistemului (4.2).
- Proiectarea și implementarea unui algoritm de serializare simplu și eficient folosit pentru transportul valorilor variabilelor de intrare și de ieșire între modulele software ce extind funcționalitățile oferite de SEA (4.2.1)
- Realizarea unei analize privind eficienta și viteza de execuție a SEA (0)
- Proiectarea și implementarea unui modul software, bazat pe SEA, ce poate fi folosit pentru interacțiunea cu operatorii umani (4.4.1)
- Proiectarea și implementarea unui modul software, bazat pe SEA, ce poate fi folosit pentru generarea unor mesaje la intervale de timp regulate (4.4.2).

În capitolul 5:

- Definirea setului de reguli de funcționare al canalelor de comunicație (CC) (5.1)
- Proiectarea și implementarea unui canal de comunicație bazat pe memorie partajata (5.2.1)

- Proiectarea și implementarea unui canal de comunicație bazat pe cozi de mesaje (5.2.2)
- Realizarea unei analize privind eficienta și viteza de execuție a CC bazate pe memorie partajata și cozi de mesaje (5.2.3)
- Proiectarea și implementarea unui canal de comunicație bazat pe un protocol simplu peste TCP/IP (5.3.1)
- Proiectarea și implementarea unui canal de comunicație bazat pe JMS (5.3.2)
- Realizarea unei analize privind eficienta și viteza de execuție a CC bazate pe un protocol simplu peste TCP/IP și JMS (5.3.3)

În capitolul 6:

- Analiza funcționarii unei ecluze navigabile și stabilirea subsistemelor pentru care se vor realiza Simulatoare Elementare și realizarea fișierului de configurație al aplicației (6.1).
- Proiectarea și implementarea unui modul software, bazat pe SEA, pentru simularea comportamentului nivelului amonte sau aval din cadrul unei ecluze navigabile (6.2)
- Proiectarea și implementarea unui modul software, bazat pe SEA,pentru simularea comportamentului porții plane din cadrul unei ecluze navigabile (6.3)
- Proiectarea și implementarea unui modul software, bazat pe SEA, pentru simularea comportamentului porții buscate din cadrul unei ecluze navigabile (6.4)
- Proiectarea și implementarea unui modul software, bazat pe SEA, pentru simularea comportamentului vanelor de umplere și de golire din cadrul unei ecluze navigabile (6.5)
- Proiectarea și implementarea unui modul software, bazat pe SEA, pentru simularea comportamentului sasului din cadrul unei ecluze navigabile (6.6)
- Proiectarea și implementarea unui modul software, bazat pe SEA, pentru simularea comportamentului unității de control și comanda din cadrul unei ecluze navigabile (6.7)
- Proiectarea și implementarea elementelor componente ale interfeței grafice a simulatorului (6.8)

Rezultatele cercetării au fost publicate pe durata realizării acestei lucrări în cadrul a șase articole, după cum urmează:

- **Lucian-Florentin Bărbulescu**, Marin Lungu, Dan-Ovidiu Andrei, *Distributed system for industrial simulation,* Analele Universității din Craiova, Seria Automatica, Calculatoare, Electronica și Mecatronica, Volumul 6(33), issue. 1, pp. 1-5, ISSN: 1841-0626, Revista CNCSIS, cat. B+, 2009
- **Lucian-Florentin Bărbulescu**, *Functional analysis of a communication framework used în a modular simulator,* 14th International Conference on System Theory and Control, Proceedings, Pp. 62-66, Sinaia, Romania, 17-19 October 2010, ISSN: 2068-046, 2010
- **Lucian-Florentin Bărbulescu**, *An algorithm designed to determine the optimum number of communication channels în a modular simulator,* Buletinul Institutului Politehnic din Iași, Automatic Control and Computer Science Section, Nr. LVII (LXI), Fasc. 3, 2011, pag. 21-32, cat. B+
- **Lucian-Florentin Bărbulescu**, *The abstract elementary simulator – the base component of a modular simulator,* 16th International Conference on System Theory, Control and Computing, Proceedings, Sinaia, Romania, 12-14 October 2012, ISBN: 978-606-834-848-3, IEEE Catalog Number CFP1236P-CDR, 2012
- **Lucian-Florentin Bărbulescu**, Viorel Mînzu, *Designing a modular simulator for a navigation lock*, The Annals of « Dunarea de Jos » University of Galati, Fascicle III, Electrotechnics, Electronics, Automatic Control, Informatics, ISSN 1221-454X, 2012, VOL. 35, NO. 1

- **Lucian-Florentin Bărbulescu**, Viorel Mînzu, *Design and implementation of an elementary simulator for the rotating gates of a navigation lock*, The 19th International Conference on Control Systems and Computer Science, Bucharest, Romania, 29-31 May 2013, Submitted paper

7.2. **Direcţii viitoare de dezvoltare**

Sistemul de simulare propus este într-o stare permanenta de actualizare. Sunt urmărite o serie de dezvoltări viitoare care sa extindă funcționalitățile existente și sa simplifice cat mai mult utilizarea sistemului.

În primul rând se dorește realizarea unei aplicații grafice cu ajutorul căreia să se poată edita cât mai simplu fișierul de configurație. Această aplicație va prezenta o lista cu componentele existente și îi va permite administratorului sa interconecteze foarte simplu (prin Drag and Drop) simulatoarele elementare pentru realizarea diverselor cheme de simulare. În plus, o componenta a acestei aplicații va permite editarea vizuala a interfeței grafice și generarea secțiunii aferente acesteia din cadrul fișierului de configurație. O posibilă evoluție o reprezintă și extragerea tuturor informațiilor legate de interfața grafică într-un fișier de configurație separat, însă această abordare trebuie în prealabil studiată pentru a se evalua impactul asupra funcționării sistemului.

O alta evoluție avută în vedere este legată de modul de funcționare al simulatorului elementar abstract, și anume modul în care sunt gestionate fluxurile de date. În acest moment modificarea valorii oricărei variabile de intrare ce are alocat un flux de date conduce automat la calculul valorilor variabilelor de ieșire aferente și transmiterea valorilor acestora, chiar dacă pentru același flux de date mai sunt alocate și alte variabile de intrare. Dacă se dorește ca fluxul de date să se declanşeze doar după primirea tuturor valorilor variabilelor de intrare aferente lui, atunci trebuie să se implementeze cod auxiliar, de către dezvoltator, în interiorul funcției de procesare a datelor. Pe viitor se dorește ca aceasta funcționalitate să fie suportată de componenta abstractă și activată prin intermediul fișierului de configurație. În plus se dorește și implementarea posibilității de a atribui unei variabile mai multe fluxuri de date. Toate aceste modificări vor avea un impact asupra timpului de execuție și de aceea noi verificări trebuie efectuate.

Implementarea de noi simulatoare elementare pentru diverse componente este de asemenea o țintă viitoare. Se dorește realizarea unei biblioteci de componente ce pot fi folosite pentru realizarea unor scheme din ce în ce mai complexe testându-se astfel și limitele sistemului. Se urmărește și proiectarea detaliată și implementarea unei componente cât mai generice ce va putea fi folosită pentru comunicația cu echipamentele reale.

Și canalele de comunicație vor suferi îmbunătățiri, atât din punct de vedere al numărului de implementări oferite, cât și din punct de vedere al modului în care gestionează datele. Se urmărește adăugarea posibilității de criptare și compresie a datelor dar se și analizează posibilitatea de a introduce o filtrare bazată pe destinație astfel încât un canal de comunicație să trimită simulatoarelor elementare doar datele strict necesare acestora.

Bibliografie

Ali-Reza Adl-Tabatabai, Michaeł Cierniak , Guei-Yuan Lueh , Vishesh M. Parikh and James M. Stichnoth, *Fast, effective code generation în a just-în-time Java compiler*, Proceedings of the ACM SIGPLAN 1998 conference on Programming language design and implementation, p.280-290, June 17-19, 1998, Montreal, Quebec, Canada

Alois Ferscha and Michael Richter, *Java based conservatice distributed simulation*, Proceedings of the 1997 Winter Simulation Conference, pp 381-388

Andrew Tanenbaum, *Distributed Operating Systems*, Prentice Hall; First edition, 1994

Averill M. Law and David W. Kelton, *Simulation Modeling and Analysis*, McGraw Hill, 1982,

Benjamin M. Brosgol, *Ada Language Suits Real-Time Safety-Critical Needs*, COTS Journal, The Journal of Military Electronics & Computing, July 2009, Available Online at: http://www.cotsjournalonline.com/articles/view/100982

Bohdan T. Kulakowski, John F. Gardner, J. Lowen Shearer, *Dynamic modeling and control od Engineering Systems*, Cambridge University Press, 2007

Bruce Eckel, *Thinking în Java, 3rd ed. Revision 4.0*, Prentice Hall, 2003

Bruce Powel Douglass, *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*, Addison Wesley, 2002

Bruce Snyder, Dejan Bosanac, and Rob Davies, *ActiveMQ în Action*, Ch. 1, Manning 2011, ISBN 1933988940,

Christopher A. Chung, *Simulation modeling handbook – A practical approach*, CRC Press, Inc. Boca Raton, FL, USA 2004, ISBN: 0-8493-1241-8

David R. Jefferson, *Virtual Time*, ACM Transactions on Programming Languages and Systems, vol. 7, no. 3, pp. 404–425, July 1985.

Esteban Egea López, Javier Vales Alonso, Alejandro Santos Martínez Sala, Pablo Pavón Mariño and Joan García Haro, *Simulation scalability issues în wireless sensor networks*, IEEE Communications Magazine, vol. 44-7, pp. 64-73, 2006, ISSN 0163-6804

Gabriele D'Angelo, *Parallel and Distributed Simulation from Many Cores to the Public Cloud (Extended Version)*, Proceedings of the 2011 International Conference on High Performance Computing and Simulation (HPCS 2011),Istanbul (Turkey), IEEE, July 2011. ISBN 78-1-61284-382-7

Hagit Attiya and Jennifer Welch, *Distributed Computing. Fundamentals, Simulations and Advanced Topics*, Second Edition, Wiley-Interscience, 2004

Joshua Auerbach, David F. Bacon, Bob Blainey, Perry Cheng, Michael Dawson, Mike Fulton, David Grove, Darren Hart and Mark Stoodley, *Design and implementation of a comprehensive real-time java virtual machine*, Proceedings of the 7th ACM & IEEE international conference on Embedded software, Pages 249-258, ACM New York, NY, USA 2007, ISBN: 978-1-59593-825-1

Kanianthra Mani Chandy and Jayadev Misra, *Asynchronous Distributed Simulation via a Sequence of Parallel Computations*, Communications of the ACM, vol. 24, no. 11, pp. 198–205, November 1981.

Kanianthra Mani Chandy and Jayadev Misra, *Distributed Simulation: A Case Study în Design and Verification of Distributed Progra*ms, IEEE Transactions on Software Engineering, Vol. SE-5, No. 5, IEEE Computer Society, New York, NY, USA, September 1979, pp. 440-452

Lennart Ljung and Torkel Glad, *Modeling Of Dynamic Systems*, Prentice Hall, 1994,

Louis G. Birta and Gilbert Albez, *Modeling and Simulation, Exploring Dynamic System Behavior*, Springer-Verlag, 2007

Martin Schoeberl, *A Java Processor Architecture for Embedded Real-Time Systems*, Journal of Systems Architecture Vol. 54/1--2 , pp. 265-286, 2008

Mihai Mocanu, *Sisteme dinamice cu evenimente discrete. Abordări paralele*, PhD Thesis, Diploma no. 3337, 1999

Richard M. Fujimoto, *Parallel discrete event simulation*, În Proceedings of the 21st conference on Winter simulation, WSC '89, pages 19–28, New York, NY, USA, 1989. ACM.

Richard M. Fujimoto, *Parallel discrete event simulation*., Communication of the ACM, vol. 33, no. 10, pp. 30-53, October 1990

Richard M. Fujimoto, *Parallel and Distributed Simulation Systems*, First Edition, Wiley-Interscience, 2000

Richard M. Fujimoto, Tutorial: Parallel & distributed simulation systems: from Chandy/Misra to the High Level Architecture and beyond, 2000.

Qusay H. Mahmoud, *Sockets programming în Java: A tutorial* [Online]. Available: *http://www.javaworld.com/jw-12-1996/jw-12-sockets.html*

Annotations [Online]. *Available: http://docs.oracle.com/javase/1.5.0/docs/guide/language/annotations.html*

Ecluza [Online]. *Available: http://ro.wikipedia.org/wiki/Ecluz%C4%83*

Hagen-Poiseuille equation [Online]. Available: *http://en.wikipedia.org/wiki/Poiseuille's_law*

J2SE 5.0 Performance White Paper *[Online]*. *Available:* http://www.oracle.com/technetwork/java/5-136747.html

Java Performance [Online]. Available: *http://en.wikipedia.org/wiki/Java_performance*

Java serialization algorithm revealed [Online]. Available: *http://www.javaworld.com/community/node/2915*

JNA [Online]. Available: http://today.java.net/article/2009/11/11/simplify-native-code-access-jna

Viscosity [Online]. Available: *http://en.wikipedia.org/wiki/Dynamic_viscosity*