UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DOMENIUL: Calculatoare și tehnologia informației
SPECIALIZAREA: Tehnologia informației

# Design și implementare platformă distribuită cu agenți mobili

LUCRARE DE DIPLOMĂ

Coordonator științific
Ș. l. dr. ing. Cristian Nicolae Buțincu

Absolvent
Cristina Surdu

Iași, 2019

DECLARAȚIE DE ASUMARE A AUTENTICITĂȚII
LUCRĂRII DE DIPLOMĂ

Subsemnatul(a) SURDU CRISTINA,
legitimat(ă) cu CI seria VS nr. 640117, CNP 2961804134129
autorul lucrării
DESIGN ȘI IMPLEMENTARE PLATFORMĂ DISTRIBUITĂ CU AGENȚI MOBILI

elaborată în vederea susținerii examenului de finalizare a studiilor de licență organizat de către Facultatea de Automatică și Calculatoare din cadrul Universității Tehnice „Gheorghe Asachi" din Iași, sesiunea VARĂ a anului universitar 2019, luând în considerare conținutul Art. 34 din Codul de etică universitară al Universității Tehnice „Gheorghe Asachi" din Iași (Manualul Procedurilor, UTI.POM.02 – Funcționarea Comisiei de etică universitară), declar pe proprie răspundere, că această lucrare este rezultatul propriei activități intelectuale, nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române (legea 8/1996) și a convențiilor internaționale privind drepturile de autor.

Data Semnătura

_____

# Cuprins

# Design și implementare platformă distribuită cu agenți mobili

Cristina Surdu

# Rezumat

Această lucrare are scopul de a proiecta și implementa o platformă distribuită folosind agenți mobili. Un sistem distribuit se definește ca fiind sistemul ale cărui componente sunt amplasate pe calculatoare diferite și care comunică între ele.

Caracteristicile principale ale unui agent mobil, definit ca obiect software, constă în faptul că are un mediu de execuție, știe când mediul de execuție se schimbă și acționează în concordanță cu schimbările, are control asupra propriilor acțiuni, are proprietatea de mobilitate, care îi permite să se plimbe prin rețea și are posibilitatea de a crea clone.

Unul dintre cele mai importante beneficii ale tehnologiei agentului mobil este abilitatea de a migra la diferite locații de rețea cu scopul de a accesa componentele software statice local, în detrimentul interacțiunii prin intermediul apelurilor de procedură la distanță.

Agenții se împart în două categorii, agenți mobili și agenți staționari, care aparțin unei agenții. Agenții mobili pot fi expediați de la o agenție la alta, pe când cei staționari nu au această abilitate, dar oferă informații legate de mediul de unde rulează aplicația (informații de sistem). Fiecare agent are sarcinile trasate în propriul cod, iar agenția controlează execuția fiecăruia.

Mediul de rulare a agenților este reprezentat de agenție. Clasa agenției conține metode implementate menite să ia decizii pentru un agent: crearea unui agent, mobil sau staționar, trimiterea lui către o locație dată în rețea, locație caracterizată de o adresă IP și un port, clonarea unui agent, dezactivarea sau ștergerea unui agent din cadrul contextului agenției.

Agenția va „asculta” pentru noi conexiuni cu alte agenții cu scopul de a primi agenți in contextul său și de a le oferi un nou mediu de rulare.

Pentru a se putea deplasa agentul trebuie să treacă prin procesul de serializare. Astfel el își rezumă execuția, transferând codul și datele către locația destinație unde este recreat pentru a-și continua sarcinile.

Platforma este reprezentată de noduri de rețea cu o configurație stabilită dinainte. Fiecare agenție își cunoaște nodurile vecine, și astfel cea mai importantă activitate a unui agent este parcurgerea rețelei, în mod autonom fie pentru colectarea diferitelor resurse indicate, fie pentru efectuarea unor sarcini bine stabilite.

# Introducere

Recent, calculul distribuit a cunoscut o evoluție deosebită datorită utilizării agenților mobili drept paradigmă de proiectare și programare, dotat cu capabilități inovatoare, în detrimentul sistemului de tip client-server unde aplicațiile sunt legate de anumite noduri din rețele. Aceștia au capturat interesul cercetătorilor și industriei, fiind în măsură să migreze în mod autonom de la un nod la un altul în rețea, transferând codul și datele, ceea ce le permite să le gestioneze în mod eficient, să realizeze calcule, să adune informații și să îndeplinească sarcini.

Paradigma agentului s-a transformat mai târziu în managementul mobilității, inspectând compatibilități și convergențe destul de interesante. Sistemele cu agenți mobili sunt o tehnologie destul de inovatoare care a găsit un loc în aplicații complexe, cum ar fi comerțul electronic, telecomunicațiile, gestionarea rețelelor distribuite, căutare adaptivă și personalizată de informații, jocuri pe calculator și exemplele pot continua. Datorită autonomiei, independenței, adaptabilității și în special mobilității, agenții mobili au capacitatea de a-și atinge obiectivele în mod fiabil și flexibil. Agenții mobili se mișcă de la un nod la altul în rețea, au în posesie toate resursele și își execută propriile sarcini independent de gazdele la care migrează.[1]

Agenții mobili sunt un tip special de cod mobil. Un agent mobil este un program care poate migra de la o gazdă de pornire la multe alte gazde într-o rețea de sisteme informatice cu o sarcină specificată de proprietar. Funcționează autonom și comunică cu alți agenți și sisteme gazdă. În timpul migrării, agentul poartă tot codul său și starea de execuție completă cu el.

Pentru a evidenția avantajele agentului mobil, care migrează direct la gazdă cu scopul de a accesa resursele locale, în raport cu alte paradigme care utilizează codul mobil, vom trece în vedere câteva diferențe importante.

Diferența dintre codul mobil și agenții mobili este faptul că agenții mobili inițiază procesul de migrare ei înșiși, în timp ce migrarea codului mobil se face de către alte componente. Codul mobil migrează numai de la un server la un client și nu lasă clientul să migreze înapoi la server sau la alt client. De exemplu, durata de viață a codului mobil, într-un caz concret, este legată de viața paginii web din care face parte și moare când browser-ul se termină sau se solicită o altă pagină Web. În contrast, agenții mobili migrează de obicei de mai multe ori.[2]

Scopul lucrării este de a scoate în evidență aspectele generale ale lucrului cu agenți mobili, de a oferi o viziune clară și completă a ceea ce presupune un proiect care utilizează agenți mobili, analizând avantajele și dezavantajele folosirii acestei tehnici într-o platformă distribuită. Totodată se va face o trecere în revistă succintă a modelelor existente deja pentru a puncta principalele caracteristici pe care le au agenții mobili și care trebuie respectate indiferent de natura aplicației.

În continuare se va face o analiză detaliată a ceea ce presupune proiectul, dezvoltând faza de proiectare, descriind metodele utilizate și deciziile pentru care a fost aleasă această abordare. Urmează faza de implementare propriu-zisă, în care se vor prezenta părțile componente ale proiectului împreună cu modalitatea de implementare. De asemenea, pentru exemplificare vor trebuie inserate rezultatele finale ale proiectului.

În altă ordine de idei, ideea principală de la care pornește proiectul este caracteristica de bază a unui agent mobil, și anume faptul că este capabil să se deplaseze printr-o rețea de sisteme informatice, să treacă de la nod la nod cu scopul de a-și îndeplini sarcinile.

# Capitolul 1. Agenții mobili în sistemele distribuite

Agenții mobili oferă o nouă și fascinantă paradigmă de proiectare pentru arhitectura și programarea sistemelor distribuite. La baza acestei paradigme stă „migrarea de cod” într-un sistem eterogen, care presupune comunicarea în sistemul distribuit prin schimbul de date între procesele acestuia. Agenții mobili reprezintă o metodă ideală de a descrie și implementa un sistem distribuit, ei reduc semnificativ traficul de rețea și reprezintă un mod eficient pentru reducerea stării de latență a rețelei.

## *1.1. Platforme distribuite*

### *1.1.1. Ce este un sistem distribuit?*

Literatura de specialitate a oferit numeroase definiții ale noțiunii de „sistem distribuit”, niciuna fiind pe deplin mulțumitoare. Într-o exprimare slabă se poate afirma că un sistem distribuit cuprinde o colecție de elemente de calcul autonomă care apare utilizatorului ca un sistem unic coerent.[3]

Această definiție se referă la două aspecte importante ale sistemelor distribuite. Primul aspect reflectă că un sistem distribuit este o colecție de elemente de calcul, fiecare fiind capabil de a se comporta independent unul de celălalt. Un element de calcul poate fi dispozitiv hardware sau proces software. Un al doilea aspect face referire la utilizatori (aplicații sau oameni) care consideră că se ocupă de un singur sistem.

### *1.1.2. Caracteristici principale*

Sistemele distribuite moderne pot, și adesea vor consta din toate tipurile de noduri, variind de la calculatoare foarte mari de înaltă performanță la calculatoare de dimensiuni reduse.

Principiul fundamental cum că nodurile acționează independent ar presupune că nu există niciun folos pentru care nodurile să conlucreze. În practică, nodurile sunt programate pentru a atinge obiective comune, acest lucru se realizează prin schimbul de mesaje dintre ele. Un nod reacționează la mesajele de intrare, care sunt procesate și, la rândul lui, continuă comunicarea prin transmiterea mesajelor mai departe. Sistemul cuprinde o colecție de noduri, de unde apare și noțiunea de gestionare a calității de membru a colecției. Prin urmare, comunicare are loc doar între membrii sistemului.

Cum s-a menționat mai sus, un sistem distribuit ar trebui să apară ca un sistem unic, coerent. Principalul scop este de a convinge utilizatorii că au de-a face cu un singur sistem ale cărui date și procese sunt dispersate pe o rețea de calculatoare. Acest lucru poate deveni destul de provocator, deoarece nu putem ignora faptul că un sistem distribuit reprezintă de fapt mai multe noduri, iar la un moment dat un nod poate suferi un comportament neașteptat, și anumite procese pot întâmpina probleme. Deși parțial, eșecurile sunt inerente oricărui sistem complex, în sistemele distribuite sunt dificil de ascuns. Astfel, concluzia este că sistemul ar trebui să funcționeze ca un întreg indiferent de interacțiunea dintre acesta și utilizatori.[3]

### *1.1.3. Obiectivele proiectării unui sistem distribuit*

Un obiectiv important al unui sistem distribuit este acela de a fi accesibil utilizatorilor pentru partajarea sau accesarea resurselor la distanță. Resursele pot fi date, fișiere, servicii de rețele. Aici factorul important îl reprezintă partea economică, un astfel de produs având costuri mai reduse.

Un alt obiectiv important al unui astfel de sistem este ascunderea faptului că procesele și resursele sunt distribuite fizic pe mai multe calculatoare. Cu alte cuvinte, distribuția proceselor și resurselor este transparentă utilizatorilor finali sau utilizatorilor aplicații.

Un sistem distribuit este deschis. În esență, reprezintă un sistem care oferă componente care pot fi ușor utilizate și integrate în alte sisteme. În același timp, un sistem distribuit, în sine, va consta adesea în componente care provin din alte surse.

După cum se observă în ziua de astăzi, conectivitatea la nivel mondial prin Internet a devenit comună, orice utilizator poate fi conectat cu un alt utilizator sau cu orice sursă de informație în orice moment. Având în vedere acest lucru, scalabilitatea a devenit una dintre cele mai importante obiective de proiectare pentru dezvoltatorii de sisteme distribuite.[3]

### *1.1.4. Tipuri de sisteme distribuite*

O clasă importantă de sisteme distribuie este cea utilizată pentru performanțe înalte de calcul. În calculul clusterului, hardware-ul de bază constă în colectarea de puncte de lucru similare sau PC-uri, conectate strâns prin intermediul unui serviciu a rețelei locale de mare viteză. În plus, fiecare nod rulează aceeași operație în sistem.

Situația devine diferită în cazul procesării în rețea. Acest subgrup este format din sisteme distribuite care sunt adesea construite ca o federație a sistemelor informatice, în care fiecare sistem ar putea fi diferit de domeniul administrativ și poate fi foarte diferit atunci când vine vorba de hardware-ul, software-ul și tehnologia de rețea implementată.

Din perspectiva procesării de tip rețea, următorul pas logic este externalizarea întregii infrastructuri necesară pentru aplicațiile cu o intensitate mare de calcul. În esență, este vorba de spre procesarea de tip „cloud[1]”: furnizarea facilitaților pentru a construi dinamic o infrastructură și a compune ceea ce este nevoie utilizând serviciile disponibile.

Sumarizând, există diferite tipuri de sisteme distribuite care pot fi clasificate ca fiind orientate către calculele de susținere pentru a prelucra informațiile. Sistemele de calcul distribuit, sunt de obicei utilizate pentru aplicații de înaltă performanță, adesea provenite din tehnicile de calcul paralel. Un câmp care a ieșit din procesarea paralelă a fost inițial o rețea cu o concentrare puternică asupra partajării resurselor la nivel global, ceea ce conduce la ce este cunoscut acum sub numele de „cloud”. Acest tip de procesare merge dincolo de calculul de înaltă performanță și sprijină, de asemenea, sistemele tradiționale de birouri, unde se găsesc baze de date care rulează procese importante.[3]

## *1.2. Paradigme de procesare distribuită*

Agenții mobili oferă o paradigmă puternică și uniformă pentru procesarea în rețea și pot revoluționa proiectarea și dezvoltarea sistemelor distribuite. Pentru a pune această afirmație în perspectivă, vom oferi o scurtă trecere în revistă și o comparație a trei paradigme de programare pentru calculul distribuit: client-server, cod la cerere și agenții mobili. Descrierea se va baza mai mult asupra paradigmei, ci nu asupra componentelor hardware sau software.

### *1.2.1. Paradigma client-server*

În paradigma client-server, un server pune la dispoziție un set de servicii care oferă acces la anumite resurse. Codul care implementează aceste servicii este găzduit de server, deci serverul este singurul care știe cum să acceseze resursele. Dacă clientul este interesat de o resursă găzduită de către server, el va trebuie să folosească serviciile serverului[4].

1 Cloud - „nor”, concept nou în domeniu computerelor și al informaticii, reprezentând un ansamblu distribuit de servicii de calcul.

În această paradigmă, nicio componentă nu este mobilă, cu excepția cererii clientului către server. Cererea conține de obicei numele serviciului și parametri suplimentari, dacă este cazul. Acest concept este comparabil cu un apel de procedură în limbile de programare și, prin urmare, au fost dezvoltate mai multe concepte de programare care oferă o utilizare convenabilă a conceptului client-server: apelul de procedură, invocarea unei metode la distanță[2].

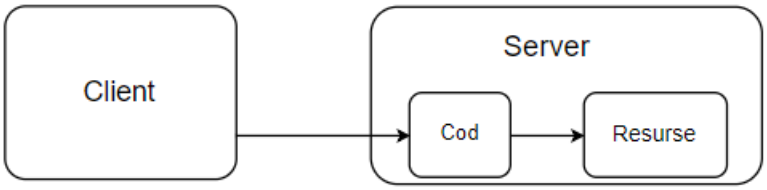


Figura 1.1. Paradigma client-server

### *1.2.2. Paradigma cod la cerere*

Conform paradigmei bazate pe cod la cerere, se obține codul atunci când este nevoie de el. Să presupunem că inițial clientul nu reușește să-și execute sarcinile, deoarece nu are cod. Din fericire, o gazdă dintr-o rețea furnizează codul necesar. Odată ce codul este primit de client, procesarea are loc. Clientul deține capacitatea procesorului, precum și resursele locale. În contradicție cu paradigma clasică client-server, clientul nu are nevoie de codul preinstalat, deoarece codul necesar va fi descărcat. Clientul are resurse și puterea de procesare, iar gazda deține codul[4].

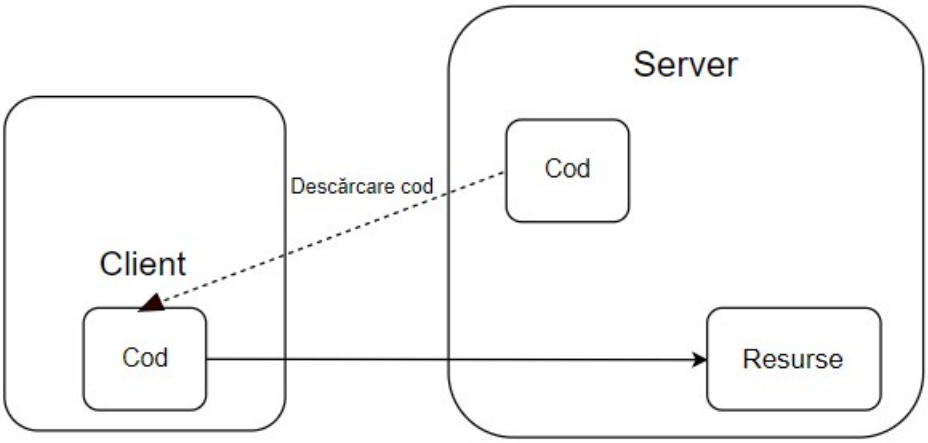


Figura 1.2. Paradigma cod la cerere

### *1.2.3. Paradigma agenților mobili*

O caracteristică cheie a paradigmei agentului mobil este aceea că orice gazdă din rețea are un grad ridicat de flexibilitate pentru a deține orice amestec de cod, resurse și putere de procesare. Capacitățile sale de procesare pot fi combinate cu resurse locale, iar codul nu este legat de o singură gazdă, ci mai degrabă este disponibil în întreaga rețea[4].

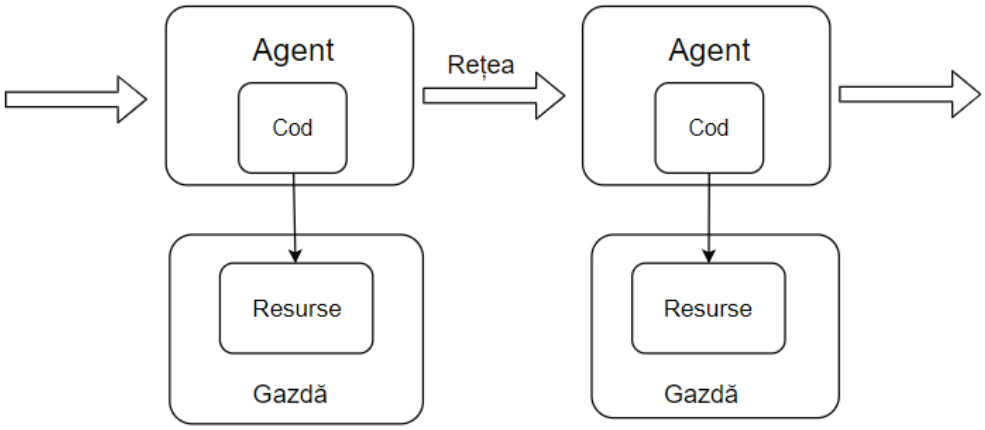


Figura 1.3. Paradigma bazată pe agenți mobili

## *1.3. Ce este un agent mobil?*

Un agent mobil reprezintă o entitate software care este lansat de proprietarul său cu o sarcină bine stabilită la un anumit nod de rețea. Acesta poate decide dacă să migreze la alte noduri din rețea în timpul rulării. Pentru migrare, agentul își păstrează datele curente, codul său și starea de execuție cu acesta. Prin urmare, este posibil să se continue executarea agentului de la platforma de destinație, exact unde a fost întreruptă înainte. Motivul migrării este, în principal, acela de a folosi resursele disponibile aflate la serverele localizate la distanță din rețea[2].

Agentul mobil reprezintă următorul salt în evoluția executabilului conținutului pe Internet, permițând transportul codului de program împreună cu informațiile de stare. Agenții mobili sunt obiecte care se pot deplasa de pe o gazdă pe alta. Când agentul se mișcă, ia codul de program, precum și datele sale.

### *1.3.1. Tipuri de agenți*

Mobilitatea este principala caracteristică a agenților, dar nu toți agenții pot fi numiți mobili. Un agent poate aparține unui nod și comunică cu împrejurimile în anumite circumstanțe, cum ar fi forma de apel la distanță sau prin mesaje. Aceștia poartă denumirea de agenți staționari.

Un agent staționar se execută numai în cadrul sistemului unde începe firul de execuție. Dacă are nevoie de informații care nu sunt accesibile sistemului, el trebuie să interacționeze cu alți agenți prin intermediul mesajelor.

Pe de altă parte, după cum este evident, un agent care are proprietatea de mobilitate este definit drept agent mobil. Acesta nu este legat de sistemul în care își începe execuția și poate călători într-un alt sistem care conține un obiect cu care agentul mobil dorește să interacționeze. El profită de acest fapt și are parte de avantajele de a fi în același nod de rețea cu obiectul respectiv pentru a obține anumite informații.[4]

### *1.3.2. Avantajele agentului mobil*

Astăzi, paradigma centrală care leagă toate tehnologiile obiectului distribuit este o

paradigmă sincronă de transmitere a mesajelor prin care toate obiectele sunt distribuite, dar staționare, și interacțiunea are loc prin intermediul schimbului de mesaje. Aceasta paradigmă este incompletă și trebuie îmbunătățită într-o oarecare măsură cu paradigme suplimentare, cum ar fi transmiterea mesajelor în mod asincron, mobilitatea obiectelor și existența obiectelor active. Agenții mobili pot oferi o viziune completă și uniformă a obiectului care poate fi procesat în mod distribuit, cuprinzând sincronizarea și asincronizarea schimbului de mesaje, existența obiectelor mobile și staționare. Agenții mobili ca elemente fizice, sunt fundamentale pentru calculul distribuit. Împreună cu mobilitatea, agenții au următoarele caracteristici unice, de asemenea, foarte importante:

1. *Agenții mobili pot reduce supraîncărcarea rețelei*. Sistemele distribuite se bazează adesea pe protocoale de comunicare care implică interacțiuni multiple pentru a realiza o anumită sarcină. Acest lucru are loc, în mod special, atunci când măsurile de securitate sunt activate, rezultând trafic major în rețea. Agenții mobilii permit împachetarea unei sarcini și expedierea acesteia către un nod de rețea, unde interacțiunile pot avea loc la nivel local. Agenții mobili sunt, de asemenea, folositori atunci când vine vorba de reducerea fluxului de date brute în rețea. În momentul în care se stochează volume foarte mari de date la gazde aflate la distanță, aceste date trebuie prelucrate local, în detrimentul transferării lor în rețea. Motto-ul este simplu: trebuie mutată procesarea în localitatea datelor, mai degrabă decât datele către locul de procesare.

2. *Agenții mobili pot depăși latența rețelei*. Sistemele critice în timp real, cum ar fi roboții în procesele de fabricație, trebuie să răspundă în timp real schimbărilor în mediul lor. Controlul acestor sisteme printr-o rețea de dimensiuni substanțiale implică latențe semnificative și astfel de latențe nu sunt acceptabile. Agenții mobili oferă o soluție, deoarece pot fi expediați de la un controler central pentru acțiune locală.

3. *Agenții mobili pot încapsula protocoale*. Atunci când datele sunt schimbate într-un sistem distribuit, fiecare gazdă deține codul care implică protocoalele necesare pentru a codifica corect datele de ieșire și, respectiv, pentru a interpreta datele primite. Cu toate acestea, pe măsură ce protocoalele evoluează, este greu de actualizat corect codul de protocol. Ca rezultat, protocoalele devin adesea o problemă. Agenții mobili, pe de altă parte, se pot muta la nodurile în rețea aflate la distanță pentru a stabili „canale” bazate pe protocoale locale.

4. *Agenții mobili se execută asincron și autonom*. Adesea, dispozitivele mobile trebuie să se bazeze pe conexiuni de rețea scumpe sau fragile. Sarcinile care necesită o conexiune continuă deschisă între un agent și o rețea fixă pot să nu fie economice sau tehnic fezabile. Pentru a rezolva această problemă, sarcinile pot fi încorporate în agenți mobili, care pot fi expediate în rețea. După expediere agenții mobili devin autonomi și asincroni.

5. *Agenții mobili se adaptează automat.* Agenții mobili au capacitatea de a-și simți mediul de execuție și de a reacționa în mod autonom la schimbare. Multi agenți mobili posedă capacitatea unică de a se distribui între gazde în rețea astfel încât să aibă o configurație optimă pentru rezolvarea unei probleme particulare.

6. *Agenții mobili sunt în mod natural eterogeni.* Procesarea în rețea este fundamentală eterogenă, atât hardware, cât și din perspectivă software. Deoarece agenții mobili sunt, în general, ajutați de calculator și sunt independenți de mediul lor de execuție, ei oferă condiții optime pentru integrarea fără probleme a sistemelor.

7. *Agenții mobili sunt caracterizați de robustețe și de toleranță la defecte.* Abilitatea

agenților mobili de a reacționa dinamic la situații și evenimente nefavorabile face mai ușoară construirea sistemelor robuste și tolerante la erori. Dacă un nod de rețea prezintă anumite defecte, toți agenții vor fi avertizați și li se va acorda timp pentru a expedia și a continua să funcționeze pe o altă rețea.[4]

## *1.4. Elementele unui sistem bazat pe agenți mobili*

### *1.4.1. Agentul mobil*

Agentul mobil are cinci atribute importante: stare, implementare, interfață, identificator. Când un agent se deplasează în rețea, el își ia aceste atribute cu el.

- *Stare*

Agentul își rezumă procesarea după expediere. Când un agent călătorește, își transportă starea cu el. Trebuie să facă acest lucru pentru a relua execuția la gazda destinație, o caracteristică a tuturor agenților mobili. Starea agentului la un moment dat este un instantaneu al execuției sale. Pentru majoritatea limbajelor de programare putem partiționa starea agentului în starea sa de execuție, contorul de programe și stiva de cadre.

Agenților nu li se cere să captureze întotdeauna și să transporte starea de execuția către gazda destinației. În multe cazuri, o aproximare este suficientă. Valorile variabilei de instanță pot ajuta agentul să determine ce trebuie să facă atunci când își reia execuția la destinație.

- *Implementare*

Ca orice alt program software, un agent mobil are nevoie de cod pentru a se executa. Când călătorește, are opțiunea de a-și lua codul sau de a merge la destinație, de a vedea ce cod există deja și de a retransmite orice cod care lipsește prin rețea (cod la cerere).

Implementarea agentului ar trebui să fie atât executabilă la gazda destinație, cât și securizată pentru executarea gazdei. Scrierea și interpretarea limbajelor independente de platformă, precum și un mediu de execuție controlat cu un mecanism de securitate restricționează accesul la resursele private ale gazdei.

- *Interfața*

Agentul are nevoie de interfață pentru comunicare. Un agent furnizează o interfață care permite altor agenți sau sisteme să interacționeze cu acesta. Această interfață poate fi orice, de la un set de metode care permite altor agenți și aplicații să le acceseze până la o interfață de schimb de mesaje.

- *Identificatorul*

Fiecare agent are un identificator care este unic în timpul vieții sale. Agenții necesită identificatori astfel încât să poată fi localizați și identificați. Deoarece identificatorul unui agent este unic global, poate fi folosit ca o cheie într-o entitate care necesită un anumit agent.[4]

### *1.4.2. Mediul de rulare – agenția*

Agentul are nevoie de un mediu în care să se execute. Definim acest mediu drept o agenție care are puterea să creeze agenți și să îi accepte în contextul său. Agenția furnizează un set de servicii pe care agentul se poate baza indiferent de locația sa specifică. Din acest punct de vedere putem privi agenția ca un sistem de operare pentru agent. Agenția dispune de atribute esențiale atunci când se dorește implementarea acesteia: interfața, resursele, locația.

- *Interfața*

Pentru a furniza anumite date despre agenție, pentru a realiza o comunicare între agent și mediul său de rulare, agenția trebuie să dispună de o interfață care să îi permită agentului să acceseze datele de care are nevoie. Numim această interfață contextul agenției și reprezentă un set de metode care furnizează anumite servicii.

- *Resursele*

Agenția oferă acces controlat la resurse locale cum ar fi rețele, baze de date, procesoare și memorii, discuri și alte tipuri de hardware, precum și servicii software. De asemenea, agenția incorporează agenții staționari creați tocmai cu acest scop, furnizarea resurselor de care agentul mobil, în călătoria sa, are nevoie.

- *Locația*

Locația este un concept important pentru agenții mobili. Definim locația unui agent de execuție ca o combinație a locului în care acesta se execută și adresa de rețea a gazdei. Această locație dorește să fie perechea adresă IP a nodului de rețea cu numărul de port.[4]

### *1.4.3. Comportamentul agentului*

Din moment ce au fost definite conceptele de bază ale unui model de agent, se poate trece la comportamentul asociat creării și transferului de agenți.

#### 1.4.3.1. Creare și eliminarea unui agent

În primul rând, trebuie examinată crearea și eliminarea unui agent din cadrul unui context. Un agent este creat într-o agenție. Crearea este inițiată de agenție, iar la instanțierea unui agent, există și posibilitatea de a i se furniza argumente de inițializare. Crearea unui agent implică două etape importante:

1. *Instanțierea și asignarea unui identificator.* Clasa de bază este încărcată, i se asociază un identificator unic, iar agentul este instanțiat și aparține agenției;
2. *Inițializare.* Există agenți care necesită anumiți parametrii de rulare. În momentul instanțierii și integrării agentului într-o agenție, el cere utilizatorului să introducă parametrii necesari execuției sale.

Decizia de eliminare a unui agent dintr-un context aparține unei agenții. Printre motivele eliminării sau dezactivării un agent dintr-un context, se numără:

- *încheierea duratei de viață.* În cazul în care agentului i s-a impus o durată de viață, în momentul în care această durată expiră, agentul va fi eliminat;
- *din motive de securitate.* Atunci când agentul depășește regulile de securitate ale sistemului unde rulează;
- *la închiderea agenției.* Odată cu închiderea unei agenții, toți agenții sunt eliminați din context.

#### 1.4.3.2. Expedierea unui agent

Procesul de transfer poate fi inițiat de către un agent însuși, de sistemul în care agentul este creat sau de sistemul în care rulează. Agentul este apoi trimis de la locul său curent și primit de către locul specificat (destinația). Agentul trebuie să cunoască destinația, odată ce acest lucru este cert, el trebuie să înștiințeze agenția că urmează să se transfere, în cazul în care ia această decizie pe cont propriu. În momentul în care se inițiază cererea de transfer, agentul trebuie să urmeze următorul traseu:

1. *Suspendarea agentului*: agentul este avertizat de un iminent transfer și trebuie să se pregătească de expediere;
2. *Serializarea agentului*: agentul împreună cu starea sa este serializată. Serializarea este procesul de creare a unei reprezentări persistente a obiectului agent care poate fi transportat prin rețea;
3. *Codificarea agentului serializat*: acest pas implică utilizarea unui protocol de transfer;
4. *Transferarea agentului*: se stabilește o conexiune în rețea cu destinația gazda și se expediază agentul codificat.

#### 1.4.3.3. Acceptarea unui agent

Primirea unui agent se face în contextul unui agenții. Aceasta ascultă pe o conexiune diferită și astfel acceptă toți agenții mobili care îi sunt destinați. Acest proces are următorii pași:

1. *Decodificarea agentului*: agenția decodează fluxul de date de intrare;
2. *Deserializarea agentului*: reprezentarea persistentă a agentului este deserializată. Clasa agentului este instanțiată, și starea agentului este refăcută.
3. *Reluarea execuției agentului*: agentul recreat este notificat de sosirea în noua destinație și își reia rularea pe un alt fir de execuție.

## *1.5. Procesarea agenților*

### *1.5.1. Fire de execuție*

Pentru a înțelege rolul firelor de execuție în sistemele distribuite și implicarea lor directă în activitatea agenților mobili, este important înțelegerea a ceea ce este un proces și cum se leagă procesele și firele de execuție. Pentru a executa a program, un sistem de operare creează un număr de procesoare virtuale, fiecare pentru a rula un program diferit. Un proces este deseori definit ca un program în execuție, adică un program care se află în prezent executat pe unul dintre procesoarele virtuale ale sistemului de operare. O problemă importantă este că sistemul de operare are mare grijă să se asigure că procesele independente nu pot afecta în mod necorespunzător sau neregulat corectitudinea comportamentului celuilalt.

Asemeni unui proces, un fir de execuție execută propria sa bucată de cod, independent de alte fire. În special, contextul unui fir de execuție constă adesea din nimic mai mult decât contextul procesorului, alături de altele alte informații pentru gestionarea firului.

O proprietate importantă a firelor de execuție este că ele pot oferi un mijloc convenabil de a permite blocarea apelurilor sistemului fără a bloca întregul proces în care firul de execuție rulează. Această proprietate determină ca firele de execuție să fie deosebit de atractive pentru utilizarea în sistemele distribuite, deoarece exprimă mult mai ușor comunicarea în formă de întreținere a mai multor conexiuni logice în același timp.[3]

În contextul, utilizării agenților mobili în sistemele distribuite, la fiecare execuție a unui agent în cadrul unei agenții, agenția creează un nou fir de execuție atașat agentului pentru o rulare independentă de firul principal de execuție. Astfel, agentul lucrează pe cont propriu detașat de alte activități întreprinse de agenție.

### *1.5.2. Procesul de serializare*

În domeniul informaticii, în contextul stocării datelor, serializarea este procesul de conversie a unui obiect într-un flux de octeți pentru stocarea obiectului respectiv și transmiterea

acestuia în memorie, într-o bază de date sau într-un fișier. Scopul său principal este de a salva starea unui obiect și pentru a putea recreat atunci când este necesar. Procesul invers se numește deserializare.

Procesul de serializare a unui obiect convertește obiectul într-un flux de date, care transportă nu doar datele, ci și informații despre tipul obiectului, cum ar fi versiunea, cultura și numele ansamblului.

Serializarea permite dezvoltatorului să salveze starea unui obiect și să îl recreeze după cum este necesar. Se pot efectua acțiuni cum ar fi trimiterea obiectului la o aplicație la distanță prin intermediul unui serviciu Web, trecerea unui obiect dintr-un domeniu în altul, trecerea unui obiect printr-un firewall ca șir XML sau menținerea securității sau a specificului utilizatorului informații în cadrul aplicațiilor[5].

O caracteristică cheie a agenților mobili este că pot fi serializați. Pentru a putea fi expediați de la o agenție la alta, agenții care au proprietatea de mobilitate se supun procesului se serializare. Forma serializată a obiectului trebuie să fie capabilă să restaureze corect starea obiectului într-o nouă instanță.

## *1.6. Exemple de platforme bazate pe agenți mobili*

Dorim să descriem câteva platforme existente pentru dezvoltarea sistemelor bazate pe agenți mobili. Vom enumera caracteristicile generale, configurație, abordările de comunicare pentru modelele IBM Aglets, platforma Grasshopper și JADE.

### IBM Aglets

Un exemplu elocvent este modelul scris în Java dezvoltat de **IBM Aglets Workbench**. Modelul se bazează pe crearea obiectelor mobile Java. Aceste obiecte poartă denumirea de *aglet,* ca o combinație între denumirile *applet* (aplicație software de dimensiuni reduse scrisă în limbajul de programare Java, folosită pentru a adăuga o anumită funcționalitate) și *agent*. [6]

IBM Aglets reprezintă una dintre primele platforme Java pentru agenții mobili. Obiectele *aglets* au capacități de migrare și comunicare, Java sprijină interacțiunile dintre agenți folosind mesaje, fluxuri de octeți și aplicarea metodei la distanță[7].

Modelul de programare Aglet este bazat pe evenimente, acesta permite programatorului să creeze „ascultători” personalizați în aglet. „Ascultătorii” vor prinde evenimente speciale în ciclul de viață al unui aglet și poate permite programatorului să ia măsuri când un aglet este expediat[4].

### Platforma Grasshopper

O altă abordare este platforma **Grasshopper,** bazată, de asemenea, pe limbajul de programare Java. Mediu său de procesare distribuit este structurat în regiuni, agenții și locuri. Regiunile oferă servicii de denumire și facilitează gestionarea componentelor distribuite. Agenții Grasshopper pot fi mobili și staționari și se găsesc în cadrul unei agenții. Fiecare gazdă din rețea este dispusă să accepte și să execute agenții, și să conducă cel puțin o agenție. Agenția reprezintă un mediu de rulare pentru agenții mobili și staționari[7].

### Modelul JADE

**JADE** (Java Agent Development Framework) este un program software implementat în totalitate în limbajul de programare Java și reprezintă un alt exemplu elocvent al utilizării agenților mobili. Această abordare simplifică implementarea sistemelor multi-agent printr-un mediu de lucru care respectă specificațiile FIPA (Foundation for Intelligent Physical Agents) și

printr-un set de instrumente grafice care suportă fazele de depanare și implementare. Un sistem bazat pe JADE poate fi distribuit între mașini, iar interfața grafică poate fi configurată la distanță. Configurația poate fi modificată chiar și la momentul executării prin mișcarea agenților de la o mașină la alta.

Pe lângă abstractizarea agentului, JADE oferă un model puternic de execuție și compoziție a task-urilor, bazat pe paradigma partajată a mesajelor asincrone, un serviciu care facilitează dezvoltarea unui sistem distribuit.[8]

# Capitolul 2. Proiectarea platformei distribuite și a agenților mobili

O etapă importantă în vederea realizării unui proiect este etapa de proiectare. Aici se stabilesc principalele componente, interacțiunea dintre acestea, definirea claselor de bază și a interfețelor. În principiu, faza de proiectare determină structura proiectului și comportamentul acestuia. Proiectul final trebuie să respecte, în linii mari, cerințele stabilite în această etapă.

## *2.1. Șabloane de proiectare*

Pentru proiectarea coerentă a unei aplicații, se pot stabili la început șabloanele de proiectare. Acest pas rezolvă problemele care pot apărea în decursul dezvoltării și implementării proiectului. Printre șabloanele regăsite în aplicație, se numără : „Master-Slave” și Iterator.

### *2.1.1. Șablonul „Master-Slave”*

Modelul „Master-Slave” este adesea folosit pentru aplicații cu mai multe fire de execuție, în care trebuie rezolvate mai multe instanțe ale aceleiași probleme. Modelul definește o schemă prin care un agent „master” poate delega o sarcină unui agent „slave”.

Există mai multe motive pentru care unii agenți, ar dori să creeze alți agenți, sclavi, și să le stabilească sarcini, unul ar fi performanța. Un agent principal poate continua să efectueze alte activități în paralel cu agentul slave.

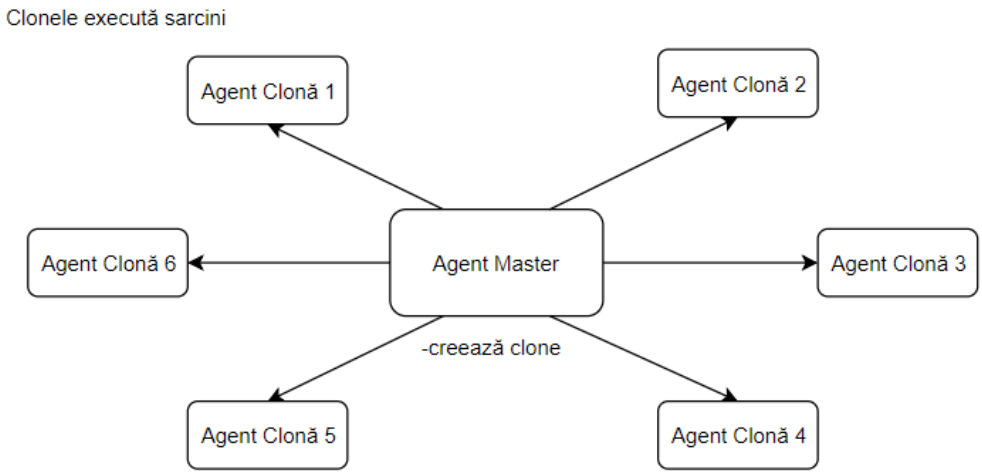


Figura 2.1. Șablonul master-slave descris în aplicație

### *2.1.2. Șablonul Iterator*

Fiind o entitate mobilă autonomă, un agent bazat pe acest model poate naviga independent la gazde multiple. Mai exact, ar trebui să poată gestiona excepții, cum ar fi gazdele necunoscute, în timp ce încearcă să se expedieze către destinații noi. S-ar putea chiar să își modifice dinamic itinerarul în funcție de informațiile locale.

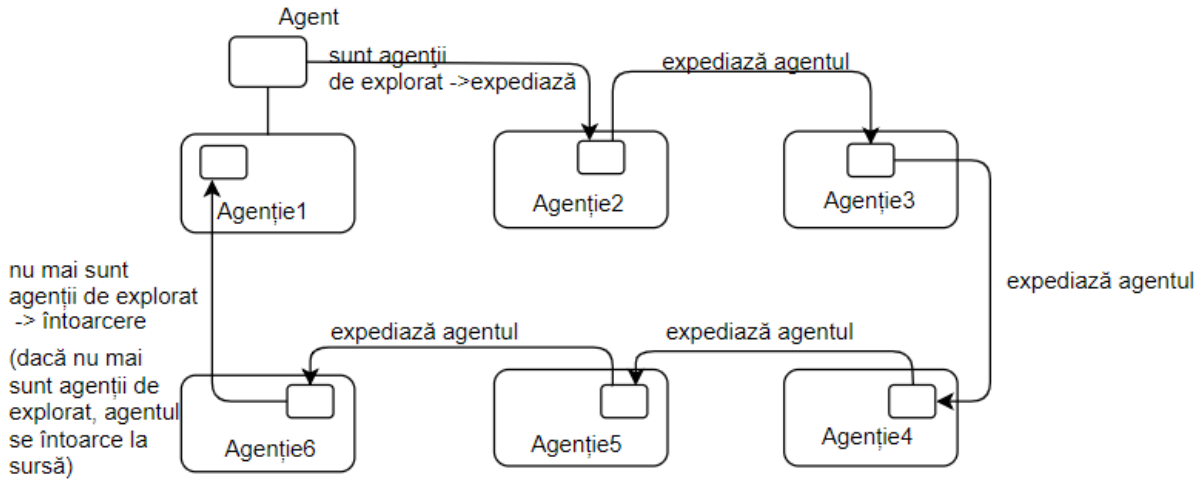


Figura 2.2. Șablonul Iterator descris în aplicație

## *2.2. Module specificate*

Aplicația este compusă din două părți: prima componentă reprezintă o librărie care integrează clasele de bază ale agentului, respectiv a agenției și interfețele fiecăruia, a doua componentă realizează interfața grafică a proiectului, mai precis interfața grafică specifică agenției.

### *2.2.1. Librăria agenților mobili*

Decizia de a include clasele de bază și interfețele într-o bibliotecă a venit din simplul motiv de a descrie o platformă distribuită în mod clar și simplu, într-o structură compactă, care conține un set de funcții care pot fi reutilizate.

#### 2.2.1.1. Clasa Agent

Clasa Agent este o clasa abstractă și desemnează obiectul pricipal de lucru, agentul mobil. Acesta încapsulează o serie de câmpuri, proprietăți și metode care îi definesc structura și comportamentul.

După cum a fost deja menționat agenții se împart în două categorii: agenți staționari și agenți mobili. Fiecare tip de agent dispune atât de o interfață comună, pentru a împărtăși metodele comune obiectului agent, cât și de o interfață separată care conține metode destinate fiecărui tip de agent. Acest lucru poate fi observat în Figura 2.3.

Modalitatea de comunicare dintre agentul staționar și agentul mobil are loc prin intermediul interfețelor definite. Agentul staționar este menit să furnizeze informațiile cerute de agentul mobil, iar agentul mobil are posibilitatea de a parcurge întreaga rețea cu scopul de accesa resursele aflate local. Prin urmare, clasa implementată este serializabilă.

Metodele agentului, preponderent, accesează membrii privați al acestuia, oferă informații

despre starea agentului, la un moment dat sau iau anumite decizii legate de activitatea agentului.

Agentul are desemnată și o durată de viață. În momentul în care o instanță nu mai realizează nicio activitate, pentru o perioadă dată de timp, el intră într-o fază de hibernare și anunță agenția cum că poate fi dezactivat. Ori de câte ori agentul va rula, perioada lui de viața va fi resetată.

Fiind o clasă abstractă, aceasta dispune și de metode abstracte în componență. Sunt metode specifice agenților mobili, cum ar fi metoda care lansează în execuție un agent și metoda care furnizează interfața grafică pentru setarea parametrilor, respectiv metoda specifică agenților staționari, cu ajutorul căreia pot fi preluate informațiile.

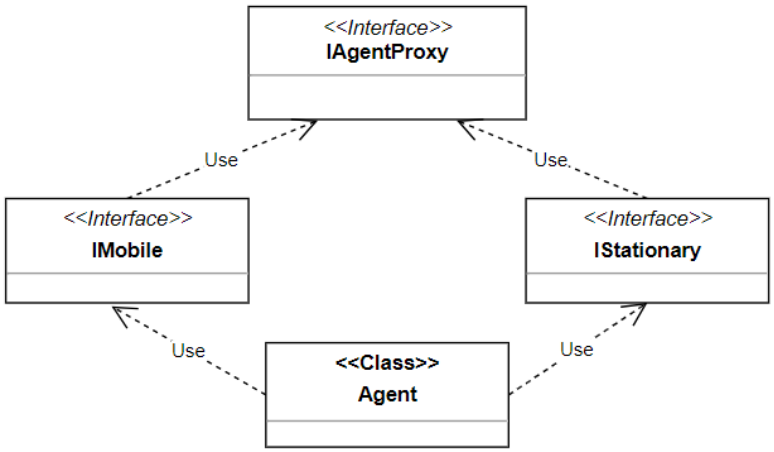


Figura 2.3. Diagrama UML prezentând relația dintre clasa Agent și interfețe.

### 2.2.1.2. Clasa Agenție

Agenția reprezintă mediul de rulare a agentului mobil. Clasa Agenție descrie acest mediu prin metodele pe care le pune la dispoziție pentru activitatea agentului. Agenția, însă, deține și un set de metode și câmpuri specifice pentru rularea proprie.

În primul rând, agenția dispune de o interfață, drept mediu de partajare a informațiilor. Toate metodele descrise în interfață sunt vizibile agentului și le accesează ori de câte ori este necesar, Figura 2.4.

La prima instanțiere a agenției, acesteia îi sunt asignați un set de agenți staționari care se ocupă de informațiile de sistem ale nodului de rețea, sau au alte întrebuințări, iar fiecare agenție ține o evidență a agenților pe care îi creează sau pe care îi găzduiește.

Agenția, după cum s-a menționat, este responsabilă de execuția agenților. Agenții mobili ajung într-o agenție atunci când sunt expediați de la o altă locație de rețea, agenția este nevoită să accepte toți agenții care inițiază acest lucru. Prin urmare, la fiecare agent nou sosit în rețea, agenția îl deserializează îi creează un nou fir de execuție, astfel conferind un nou mediu de rulare agentului.

De asemenea, de fiecare dată când agentul va fi trimis la o altă locație de rețea, el va cere agenției să îl expedieze, în acest mod se va verifica și dacă agenția destinație este sau nu conectată și se vor evita erorile de serializare.

Agenția dispune și de un set de evenimente care indică următoarele: dacă o agenție cu care se dorește o conexiune nu este disponibilă, dacă un agent este expediat dintr-o agenție, sau dacă un agent sosește într-o agenție. Evenimentele se folosesc pentru a putea fi propagate în interfața grafică.

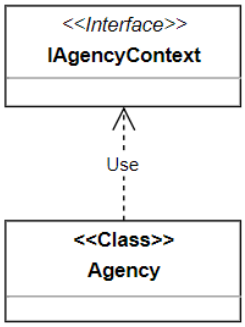


Figura 2.4. Diagrama UML prezentând agenția împreună cu interfața sa.

#### 2.2.1.3. Clase speciale pentru evenimente și excepții

Pentru a oferi informații în timp real utilizatorului și pentru a ține o evidență clară a evenimentelor care au loc în platforma distribuită, sunt implementate clase proprii care oferă tipuri de argumente pentru mai multe cazuri de evenimente.

Astfel, sunt create clase de argumente pentru evenimente de tip: agenție neconectată, mobilitatea agenților (expediere, sosire), persistența agenților (agenți activi, agenți inactivi), clonarea agenților.

Pe lângă excepțiile oferite de limbajul de programare, au fost implementate noi clase conținând excepții în concordanță cu cerințele proiectului. Excepțiile noi create au rolul de a lucra cu obiectele aplicației, agenția și agentul, pentru a arăta utilizatorului când o acțiune nu s-a desfășurat conform planului. Printre excepțiile implementate, sunt excepții care anunță că o agenție nu se găsește în cadrul topologiei de rețea, respectiv un agent nu se găsește în cadrul agenților creați pentru anumite sarcini stabilite sau că un agent nu are posibilitatea de a crea clone.

### *2.2.2. Modulul de interfață cu utilizatorul*

Pentru a pune în evidență mobilitatea agenților într-o platformă distribuită, este necesar crearea unei interfețe, unde utilizatorul poate interacționa direct cu aplicația.

Cum am menționat mai sus, responsabila de primul impuls al agenților este agenția, iar din acest motiv interfața principală aparține acesteia.

#### 2.2.2.1. Fișierul de configurare

Pentru a rula, o agenție are nevoie să cunoască cel puțin doi parametrii: adresa IP (Protocolul de Internet) și numărul portului. Dat fiind faptul că pentru a accepta agenți, agenția are comportamentul unui server, la prima creare a unei instanțe, agenția începe să „asculte”.

La fiecare rulare, agenția își ia parametrii din fișierul de configurare, potriviți stației unde rulează. În acest mod, comunicarea poate avea loc fără probleme.

Din același fișier de configurare, fiecare agenție își extrage nodurile de rețea vecine, asemeni unor liste de adiacență. În acest mod se creează o structură topologică de tip graf.

#### 2.2.2.2. Interfața agenției

Agenția are o interfață intuitivă oricărui utilizator, permite efectuarea oricărei acțiuni asupra agentului. Însă toată această funcționalitate este preluată din clasele de bază. Interfața face legătura între codul propriu zis și tot ceea ce este afișat utilizatorului. De asemenea, această componentă are acces la fișierul de configurare, și la lista statică a tipurilor de agenți care sunt disponibili pentru a fi creați. Această listă conține o evidență a agenților derivați din clasa de bază, pot fi manipulați de agenție și trimiși în execuție.

Există mai multe de tipuri de agenți implementați: agenți care parcurg întreaga rețea, atât timp cât agenția îi poate găzdui, agenți care sunt expediați doar la o anumită locație din rețea, sau agenți de tip „bumerang", ei sunt expediați la o locație dată, dar se întorc la agenția creatoare.

Interfața folosește obiectul agenției, de unde pot fi luate hotărâri precum: crearea unui agent, expedierea lui în rețea, expedierea lui către o locație dată, clonarea unui agent, dezactivarea și reactivarea unui agent sau afișarea unor anumite informații pentru un agent dat.

Figura 2.5 redă interacțiunea utilizatorului cu interfața. În diagrama am integrat și agentul, drept un actor, deoarece el poate lua decizii cu privire la crearea clonelor și expedierea acestora.

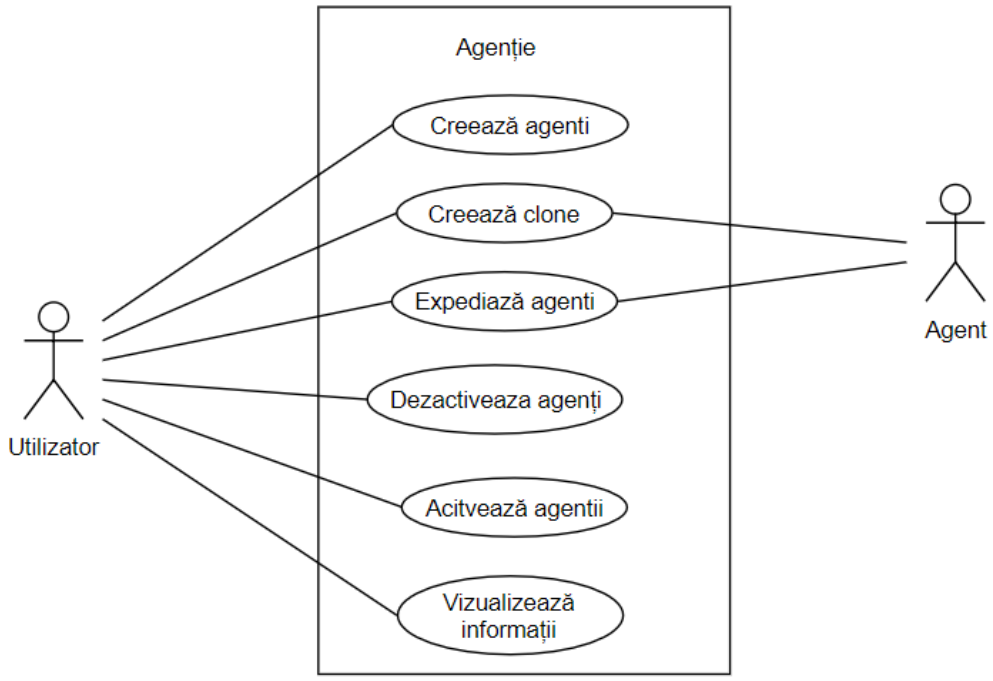


Figura 2.5. Diagrama cazurilor de utilizare

## *2.3. Interacțiunea dintre module*

Principala funcționalitate a proiectului se găsește în clasele de bază din librărie. Interfața interceptează cereri, iar pe baza acestor cereri, sunt extrase informații sau au loc anumite acțiuni în cadrul aplicației.

Mai jos este descrisă o schemă de componente care redă, în ansamblu, comunicarea între componentele proiectului.

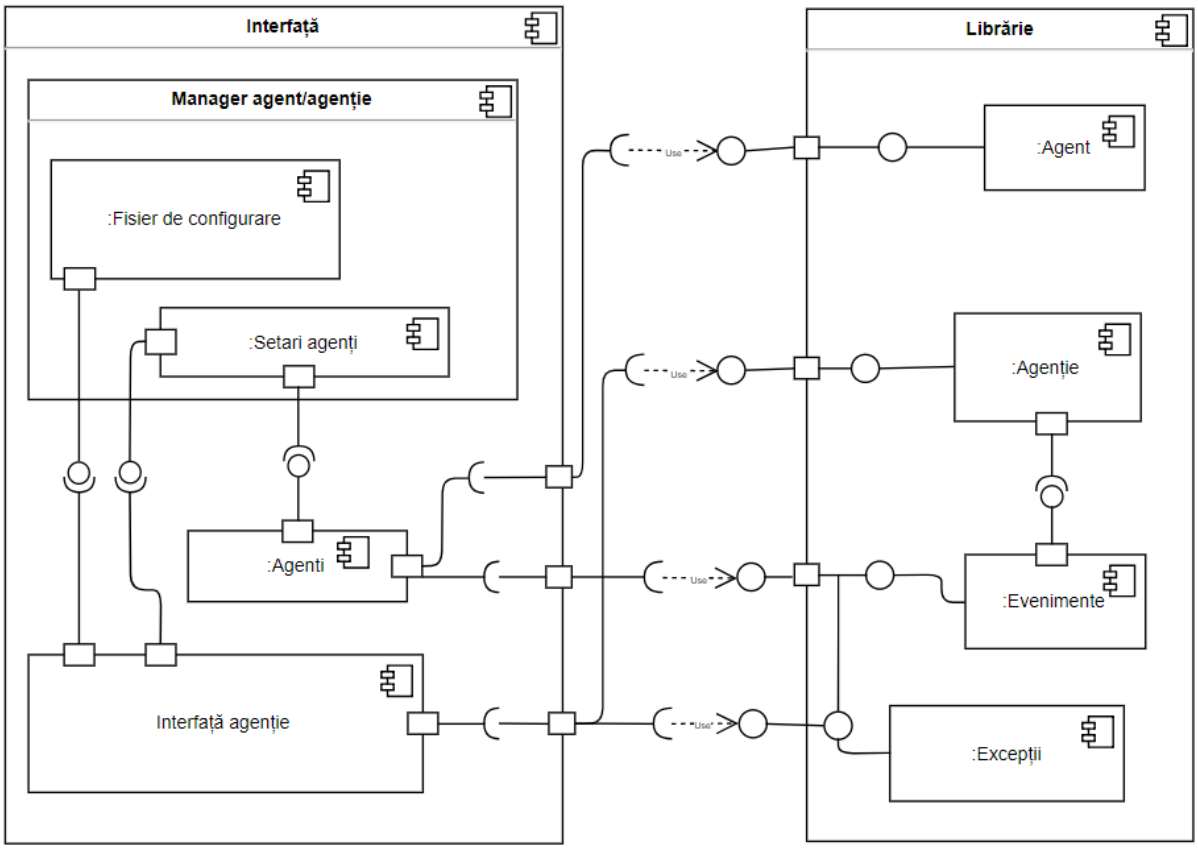


Figura 2.6. Diagrama de componente a proiectului

Fiecare componentă necesită sau asigură o interfață. Pe baza acestui aspect, a fost creată diagrama UML[2], tocmai pentru a sublinia faptul că există o comunicare strânsă între componentele proiectului. De exemplu, interfața agenției pe lângă obiectul agenție, are nevoie de fișierul de configurare, de setările agenților și de excepții.

## *2.4. Ciclul de viață al unui agent*

Agentul, reprezintă obiectul principal de lucru în platforma distribuită. El furnizează informații, adună informații sau execută sarcini. Fiind un obiect de uz, de care o agenție se poate dispensa, el trece prin mai multe stări, cu o traiectorie bine stabilită.

Fiecare stare este relevantă, deoarece compune ciclul de viață al agentului. Atât el însuși, cât și agenția sunt responsabili de stările pe care le traversează.

- *Creare.* Decizia de a crea un agent aparține unui utilizator, iar agenția onorează cererea.
- *Clonare*. Pentru a clona un agent există două posibilități: din interfața grafică a agenției sau direct de către un agent, atunci când dorește să efectueze sarcini în paralel.
- *Suspendare.* Suspendarea are loc în momentul când agentului i se termină durata de activitate sau când decide utilizatorul că dezactivarea trebuie să aibă loc.
- *Reactivare*. Pentru reactivare se folosește interfața grafică a agenției.
- *Expediere*. Se face cu ajutorul metodei implementate în clasa agenției, însa cererea poate proveni atât de la agent, cat și de la mediul unde rulează agentul (agenția).
- *Execuție*. Are loc de fiecare dată când un agent este primit într-o agenție, unde se creează

2 UML (eng. Unified Modeling Language) - limbaj standard pentru descrierea de modele și specificații pentru software

noul fir de execuție atașat.

- *Întoarcere la sursă.* Pentru agenții care se „plimbă” prin rețea, după terminarea sarcinii indicate, ei se întorc la sursă. Același traseu îl au și agenții de tip „bumerang”.
- *Eliminare*. Un agent este eliminat atunci când el este șters din lista agenților atașați agenției, sau în momentul în care o agenție este deconectată.

Figura 2.7 prezintă ciclul de viață al unui agent, de la creare până la eliminarea acestuia.

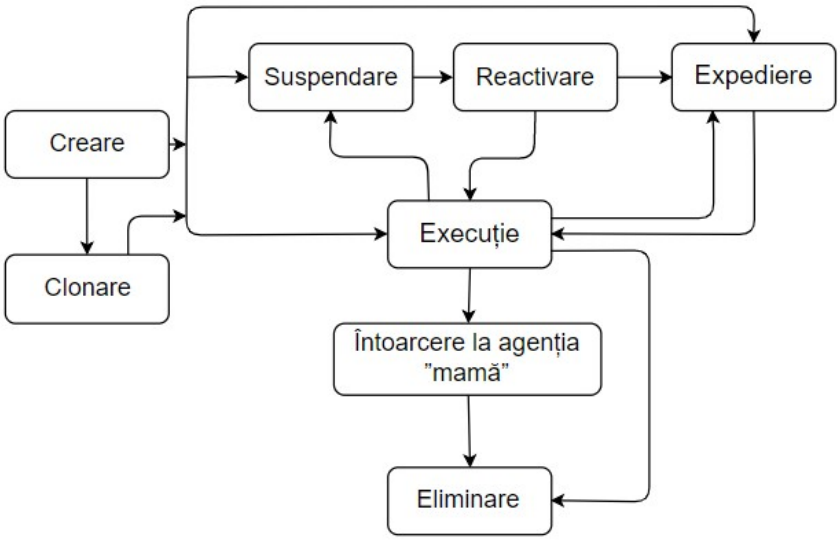


Figura 2.7. Ciclul de viață al unui agent

## 2.5. Diagrame UML

### 2.5.1. Diagrama UML de activitate

Diagrama de activitate din Figura 2.8 redă acțiunile importante din aplicație. Deși, odată rulată o instanță a agenției, ea poate fi deconectată în orice moment, relevant este ca agenția să își încheie execuția în momentul când cel puțin un agent este lansat în execuție. În acest mod, se va testa validitatea platformei distribuite implementate. În momentul în care a rulat cel puțin un agent, agenția poate fi închisă sau poate continua gestionarea platformei distribuite.

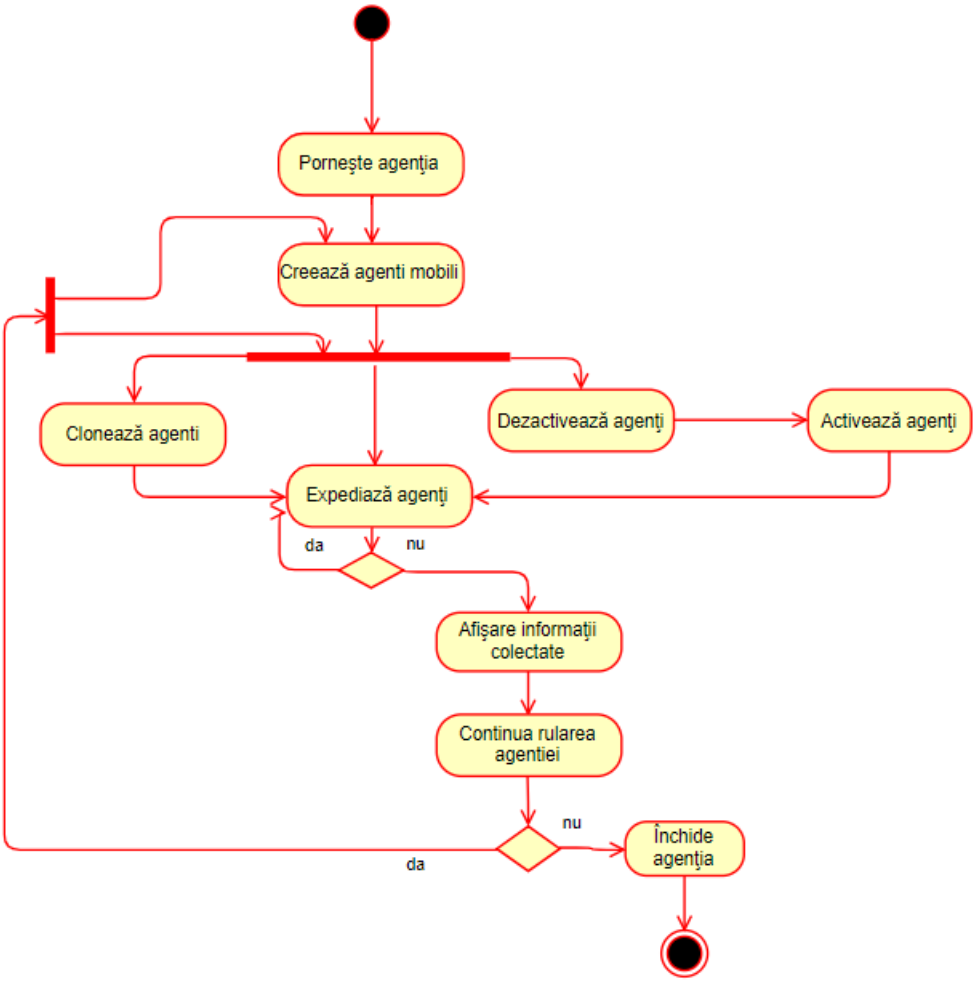


Figura 2.8. Diagrama de activitate a aplicației

### *2.5.2. Diagrama UML de clase*

Diagrama de clase se găsește la Anexa1. Va fi prezentată pe componente conform realizării proiectului, o viziune mai clară a relațiilor dintre clase reieșind și din diagrama de componente (Figura 2.6).

# Capitolul 3. Implementarea agenției și a agenților în limbajul de programare C#

## *3.1. Limbajul de programare C#: concepte, avantaje*

În istoria sa scurtă, tehnologia Microsoft .NET a devenit rapid o platformă de programare populară dezvoltând aplicații pentru stațiile de lucru și serverele Microsoft Windows. Deși cea mai mare parte a atenției mass-media au avut-o aplicațiile Web, există multe alte caracteristici care sunt utile pentru programatori de Windows.

Una dintre aceste caracteristici este noul limbaj de programare C#, dezvoltat special pentru .NET. C# devine o platformă de programare folosită pe scară largă pentru programatorii care doresc să creeze atât aplicații de rețea, cât și aplicații pentru sisteme Windows. Limbajul oferă multe resurse pentru a ajuta la crearea unor aplicații robuste. Mulți programatori migrează în limbajul C# pentru a profita de aceste resurse.[9]

### Programarea în rețea

Protocolul de Internet (IP) se află în centrul programării în rețea. IP este vehiculul care transportă date între sisteme în cadrul unui mediu de rețea locală (LAN) sau într-un mediu de rețea extinsă (WAN).

IP oferă cea mai robustă tehnică pentru trimiterea de date între dispozitivele de rețea, mai ales dacă acestea sunt situate pe Internet. Programarea folosind IP este adesea un proces complicat. Există mulți factori care trebuie luați în considerare rețeaua, tipul de rețea, congestia rețelei și eroarea în rețea[9].

Pentru programarea în rețea se folosesc socket-uri, descriptori specifici de fișiere utilizați pentru a face referire la conexiunile de rețea. Un socket definește următoarele:

- un domeniu de comunicare specific, cum ar fi o conexiune la rețea;
- un tip specific de comunicare, cum ar fi fluxul de bytes sau datagrama;
- un protocol specific, TCP (Protocol de Control al Transmisiei) sau UDP (Protocolul Datagramelor Utilizator).

După ce socket-ul a fost creat, el trebuie legat la o adresă specifică din rețea și la un port. Odată ce a fost conectat, el poate fi folosit pentru a primit și trimite date.

## *3.2. Modalități de implementare și metode folosite*

### *3.2.1. Serializarea agenților*

Agenții sunt implementați în clase serializabile care le oferă posibilitatea să circule în rețea. Mobilitatea agenților are loc în două etape:

1. *Expedierea unui agent mobil.*

   Codul sursă care trimite agentul în diferite locații din rețea se găsește în clasa agenției. Pentru a se putea deplasa agentul trebuie să treacă prin procesul de serializare. Se folosește serializarea binară, BinaryFormater. Stream-ul folosit pentru serializare este NetworkStream, ceea ce permite transmiterea fluxurilor de date prin intermediul socket-urilor în mod blocant. Pentru a instanția clasa NetworkStream cu ajutorul socket-ului este necesar ca acesta să fie conectat. Se folosește un socket cu o conexiune TCP, de tip Stream, datele transmise fiind fluxuri de bytes, bidirecționale. Un socket de acest tip comunică cu un singur participant și necesită o conexiune la distanță înainte de a începe

comunicarea.

2. *Acceptarea unui agent mobil.*

   Agenția va „asculta” prin intermediul unui socket de tip server. Acesta deservește agenților care urmează să sosească în agenție. Se folosește, de asemenea, un socket cu o conexiune TCP, de tip Stream. După conectare, agenția va „asculta” continuu pentru a realiza noi conexiuni. Odată acceptată o conexiune, metoda Accept aplicată socket-ului de tip server, returnează tot un socket, cu ajutorul căruia se realizează deserializarea agentului sosit în agenție. Pentru deserializare se folosește o instanță a clasei NetworkStream, parametrul fiind socket-ul returnat.

### *3.2.2. Tipuri de agenți implementați*

#### Agentul care străbate structura topologică de tip graf neorientat a rețelei

Căutarea în lățime într-un graf reprezintă un algoritm care este folosit pentru a căuta o singură componentă grafică conectată. Se descrie o modalitate deterministă de a explora vârfurile componentei grafice. Definiția standard a algoritmului implică o buclă iterativă peste o coadă de vârfuri[10]. Explorarea unui graf se face prin descoperirea unui vârf (numit rădăcina), se explorează mai întâi nodurile vecine acestuia, înainte de a trece la vecinii de pe nivelul următor (vecinii vecinilor). Se repetă pașii până când toate nodurile sunt vizitate.

Acest algoritm, ușor modificat, dar care se bazează pe același principiu, este încapsulat în codul agenților care sunt destinați să parcurgă rețeaua în vederea efectuării sarcinilor. Varianta folosită în proiect este cea iterativă.

Pseudocodul din Figura 3.1 redă algoritmul de căutare în lățime care se regăsește în agenții care străbat toată topologia de rețea.

```
Căutare_în_lățime(rețea, prim_nod)
        adaugă în Q (prim_nod)
        marchează prim_nod  vizitat
        cât timp (Q nu este goală) // rețeaua este parcursă în întregime
                scoate din coadă  element
                pentru fiecare vecin aparține element
                        dacă vecin este nevizitat
                                adaugă în coadă( vecin)
                                marchează vecin vizitat
```

Figura 3.1. Pseudocodul algoritmului de parcurgerea în lățime a unui graf

În coada **Q** se rețin denumirile nodurilor de rețea, iar algoritmul se termină când nu mai sunt noduri de rețea de explorat. Se începe de la nodul de unde se expediază agentul.

Acest tip de agent folosește drept metodă de parcurgere a rețelei, algoritmul de căutarea

în lățime într-un graf. În acest mod se asigură că absolut toate nodurile sunt explorate, atât timp cât nodurile sunt conectate între ele.

Cu scopul de a realiza acest lucru, agentul folosește o coadă specială, care reține, sub forma unei structuri de tip tuplă, perechea formată din nodurile pe care urmează să le străbată și drumul înapoi la sursă. În cazul în care următorul nod de rețea pe care trebuie să îl exploreze agentul nu se află în vecinătatea nodului gazdă, el parcurge drumul înapoi până când are posibilitatea de a stabili o conexiune directă cu locația destinație.

De asemenea, există posibilitatea ca un nod de rețea, să nu cunoască cu certitudine dacă agențiile vecine rulează și dacă au posibilitatea de a accepta agenți. Fiindcă nu se poate realiza comunicarea, nodul de rețea este scos din coadă și se continuă explorarea rețelei. Pentru a ține o evidență a nodurilor vizitate, agentul folosește o listă cu numele agențiilor vizitate.

În continuare, se ia pentru exemplificare un graf oarecare, și se observă traseul parcurs de un agent în momentul în care este lansat în execuție în rețea. Topologia este ilustrată în figura de mai jos.

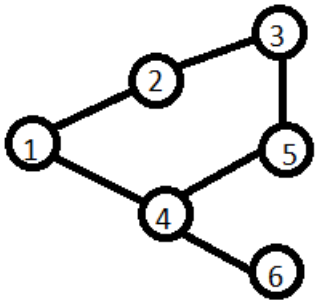


Figura 3.2. Exemplu de topologie de rețea

În Figura 3.3 și Figura 3.5 sunt ilustrați pașii pentru parcurgerea agentului în rețea, nodul de pornire fiind nodul 1.

Pasul intermediar implică întoarcerea agentului la sursă și continuarea drumului către nodul destinatar. Acest lucru se realizează deoarece nu există o conexiune directă între nodul aflat la prelucrare și nodul care urmează să fie scos din coadă. Codul sursă poate fi vizualizat în Anexa2 .

### Agentul care străbate structura topologică de tip stea a rețelei

Pentru a parcurge acest tip de topologie, agentul își creează clone, deoarece această strucură permite realizarea, cu ușurință, a paralelismului. Agentul master, aflat în nodul central creează clone, pe care le trimite către celelalte noduri. Clonele se execută și preiau informațiile, dacă este cazul, și se întorc la nodul central pentru a-i comunica master-ului rezultatele.

Figura 3.4 redă, sub formă schematică, abilitatea agentului de a crea clone și de a le expedia către celelalte noduri. După execuție, agentul se întoarce la agenția care găzduiește agentul master. Codul sursă poate fi vizualizat în Anexa2.

În topologia creată pe baza fișierului de configurare, acest tip de agent colectează resursele nodurilor vecine.

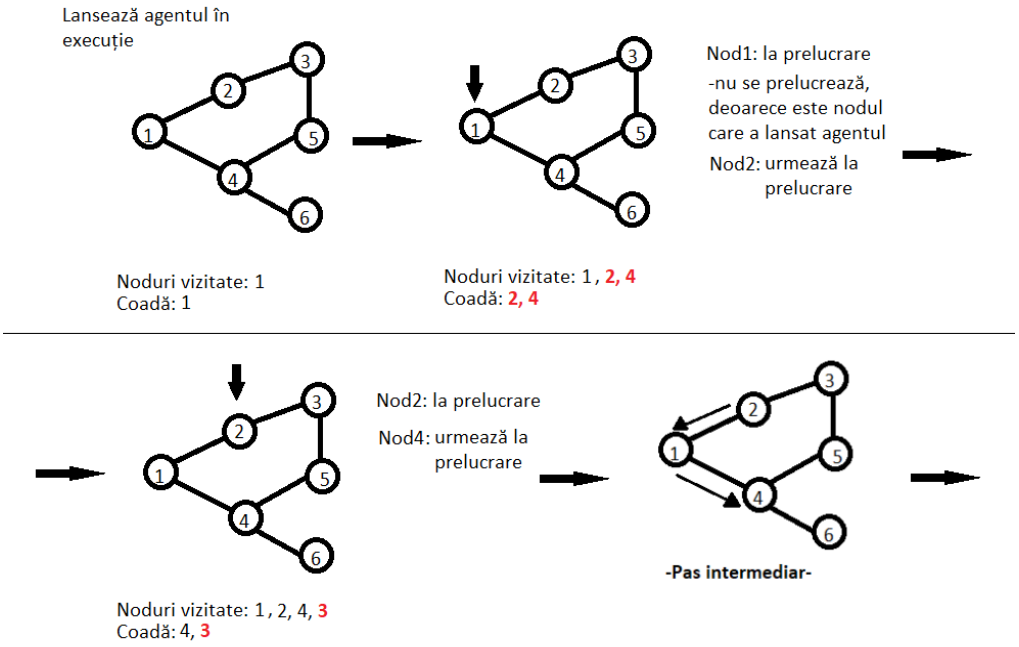


Figura 3.3. Pașii parcurși de agent în rețea

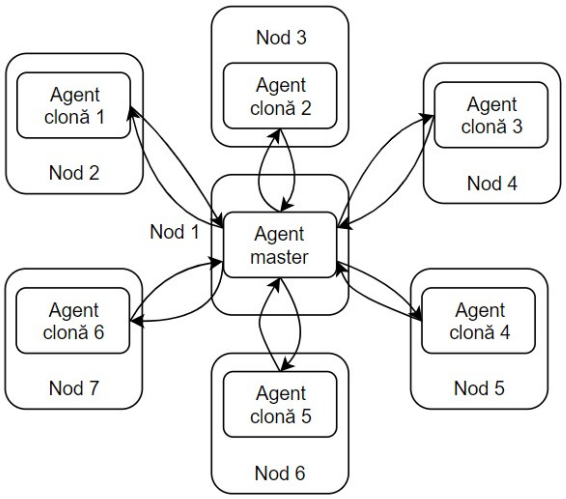


Figura 3.4. Clonarea agenților într-o topologie de tip stea

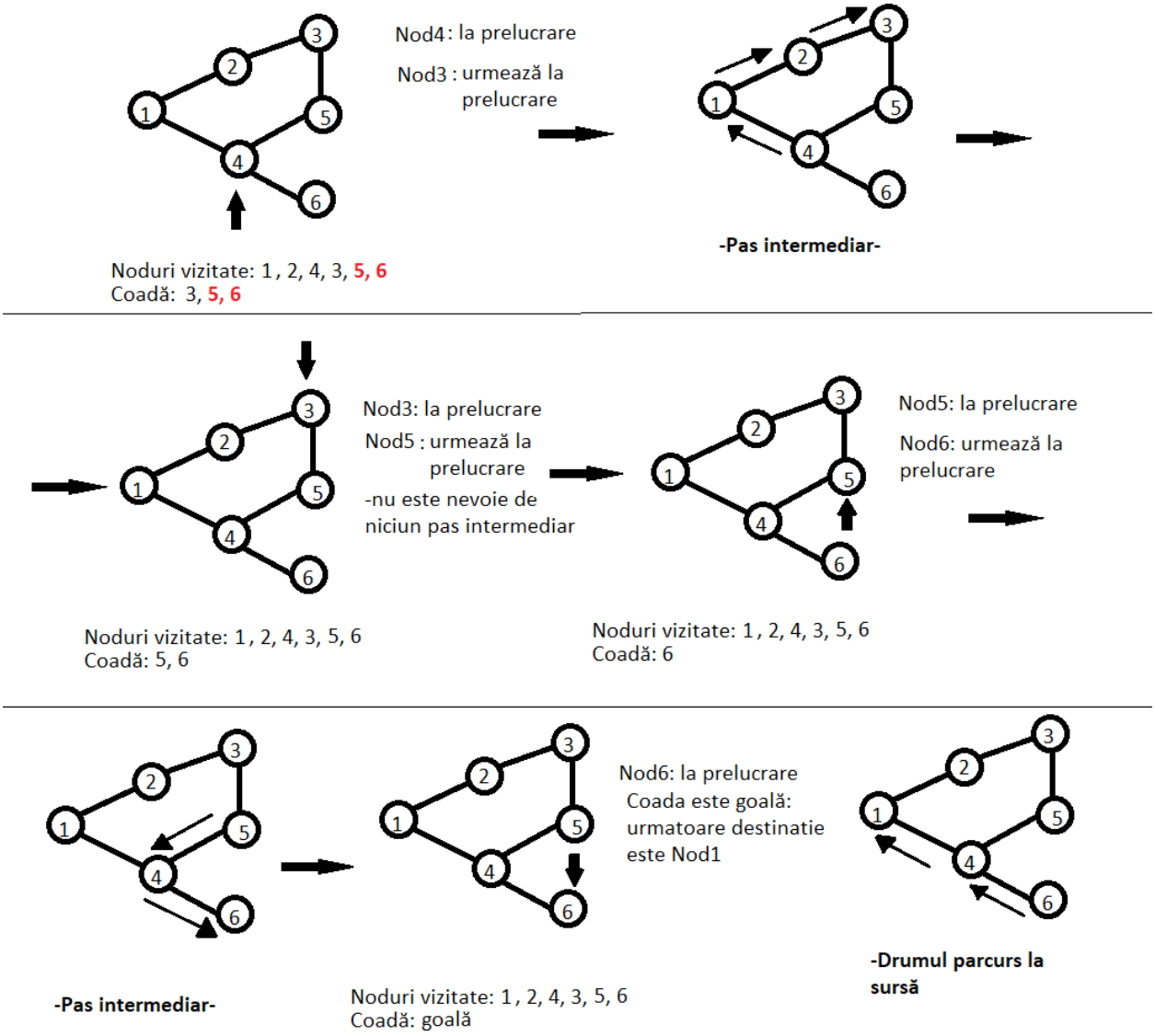


Figura 3.5. Pașii parcurși de agent în rețea (continuare)

### *3.3. Probleme întâmpinate și modalități de rezolvare*

Una din problemele întâmpinate a fost în momentul serializării și deserializării agentului împreună cu pornirea firelor de execuție. După încercări succesive eșuate, în vederea serializării, am convenit că ar fi o modalitate simplă de a folosi stream-ul de tip rețea, anume clasa NetworkStream și serializarea binară, cu ajutorul clase din C#, BinaryFormatter. Pentru sosirea agenților în rețea, am optat pentru creerea unui socket atașat agenției de tip sincron, care blochează sosirea mai multor agenți simultan în agenție, și astfel se creează o coadă de așteptare, unde fiecare agent va fi deserializat conform timpului de sosire.

A doua problemă întâmpinată a fost parcurgerea agentului în topologia de rețea graf neorientat, cu întoarcere înapoi, în momentul când nu există o legătură directă între nodul de prelucrat și nodul pe care urmează la prelucrare conform algoritmului de căutare în lățime aplicat. Pentru a depăși impasul, am păstrat în starea agentului mobil, o serie de structuri sau colecții care au furnizat agentului în fiecare locație din rețea datele necesare desfășurării sarcinii sale.

# Capitolul 4. Testarea aplicației și rezultate experimentale

Platforma distribuită proiectată este compusă din instanțele agențiile care rulează pe calculatoare diferite, utilizând adresa IP a rețelei locale de Internet și un număr de port. La rularea unei agenții pe un calculator i se va furniza numărul stației unde rulează.

## *4.1. Rularea agenției*

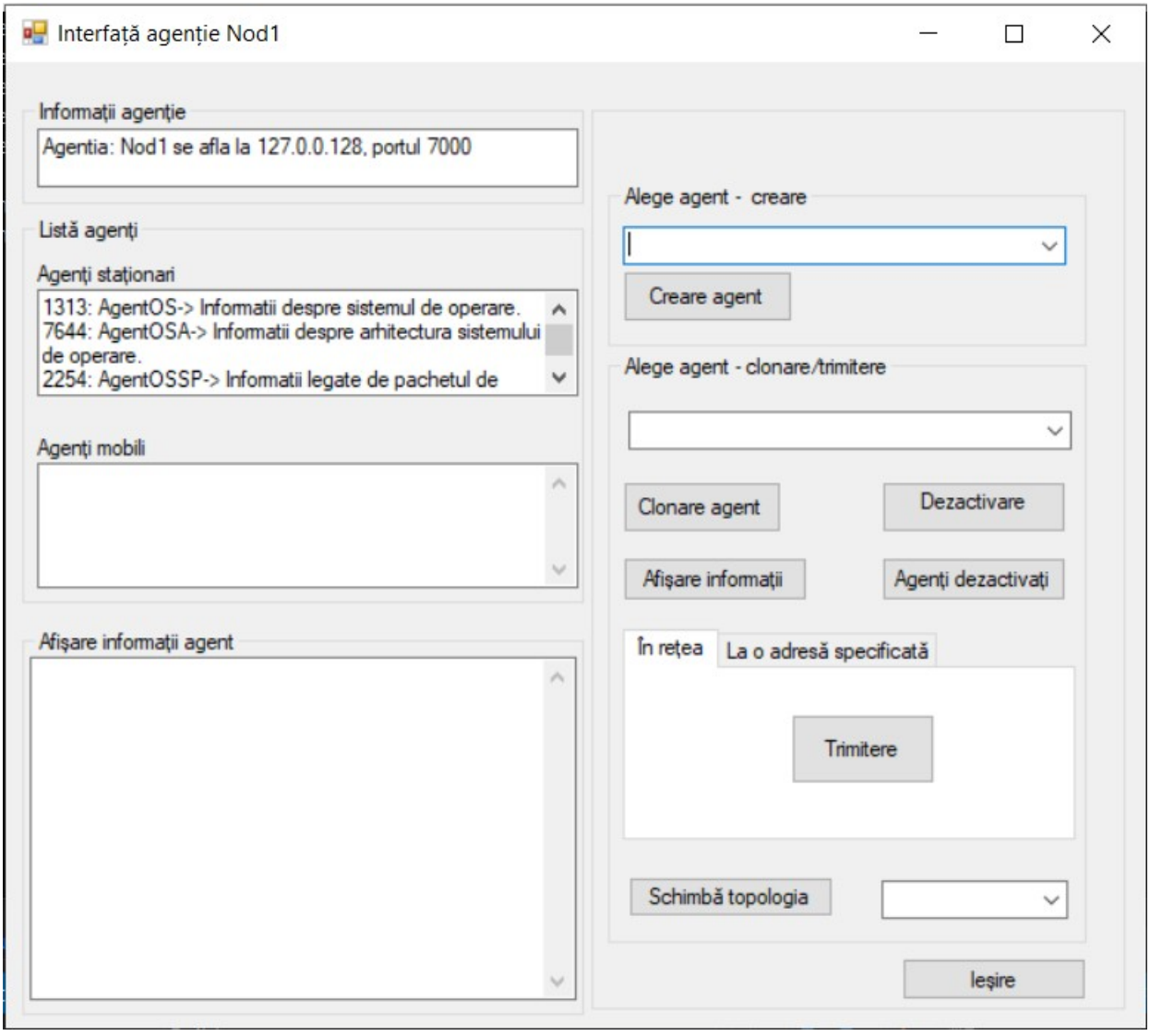


Figura 4.1. Interfața agenției

Pentru a rula o agenție este necesar doar lansarea în execuție a programului. Agenția se va conecta la adresa IP și numărul de port disponibile în fișierul de configurare și astfel devine pregătită să creeze agenți dintr-o lista dată și să îi lanseze în execuție.

Interfața conține în partea stângă chenare pentru afișarea informațiilor și în partea dreapta un panel cu butoane pentru funcționalitatea aplicației.

## *4.2. Testarea platformei utilizând agenții implementați*

Pentru a testa platforma, au fost implementați un set se agenți mobili, alocați într-o listă

statică, care pot fi vizualizați și în interfața grafică. Fiecare agent creat deține o anumită funcționalitate integrată în propriul cod.

Pentru exemplificare am rulat mai multe instanțe ale agenției, am creat un agent pe care îl voi expedia în rețea. Au fost testate diferite cazuri pentru topologia dată:

- toate agențiile sunt conectate;
- numai o parte din agenții sunt conectate (există și aici cazuri diferite);
- o singură agenție conectată.

Se ia cazul agentului care parcurge rețeaua cu mai puține noduri conectate din cate se găsesc in fișierul de configurare.

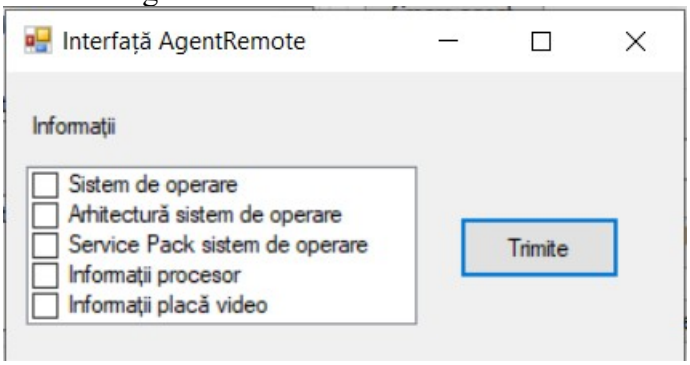


Figura 4.2: Interfața grafică atașată agentului AgentRemote

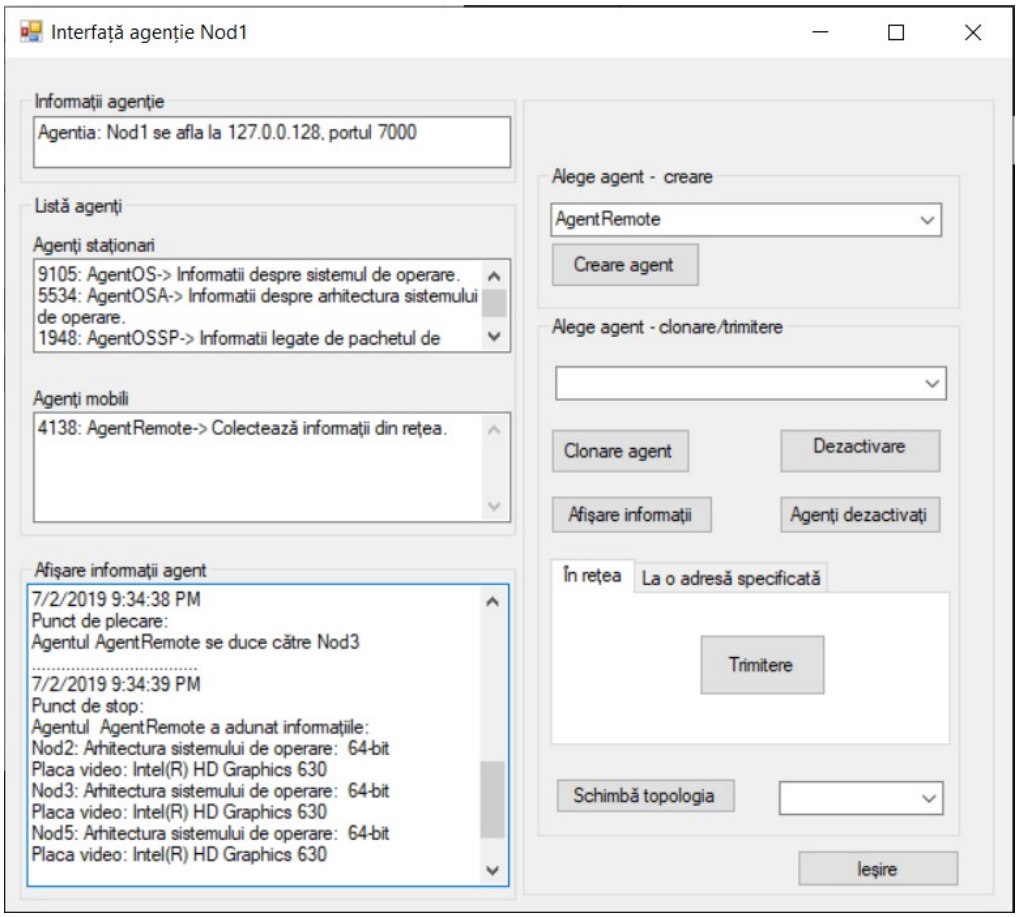


Figura 4.3: Interfața agenției care a trimis agentul în rețea

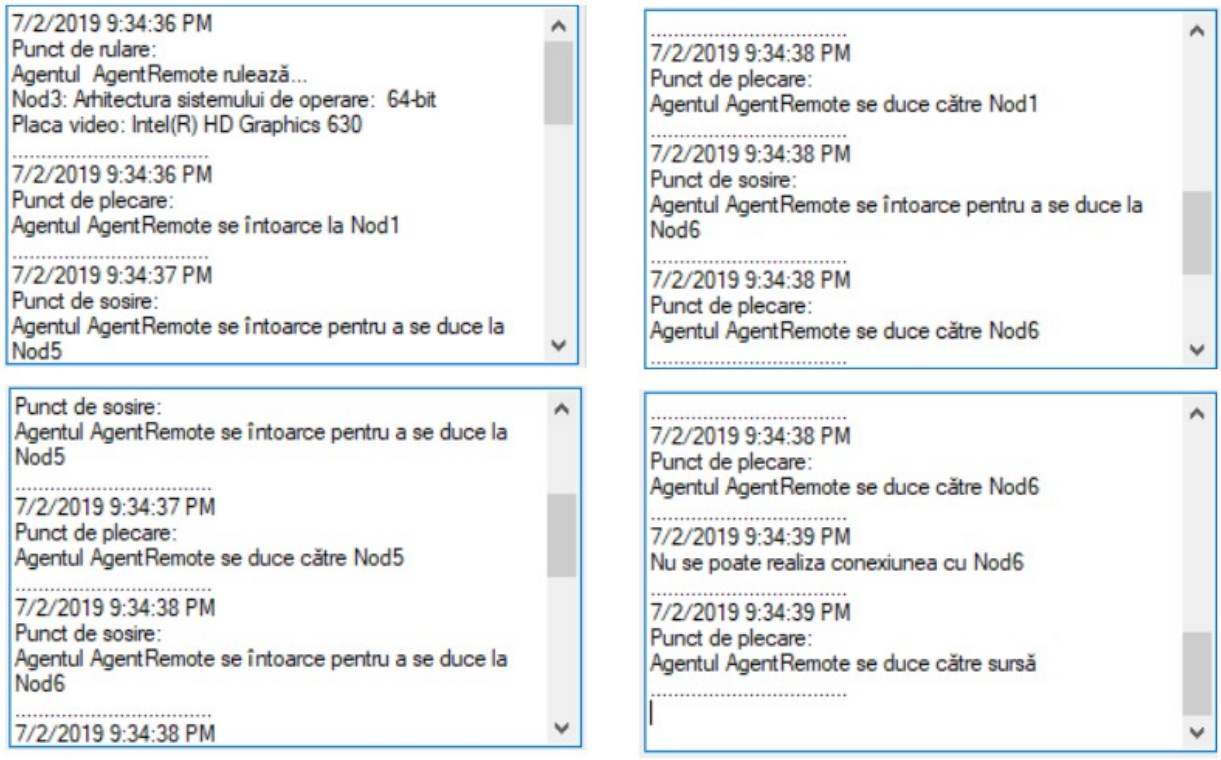


Figura 4.4: Capturi de ecran pentru consola agenției Nod3

Toate evenimentele importante precum plecarea, sosirea unui agent într-o agenție, clonarea, dezactivarea sau reactivarea unui agent sunt afișate utilizatorului în chenarul de text.

Pe lângă agentul care colectează informații este implementat și agentul care rezervă o agenție pentru o perioadă dată. El străbate rețeaua, verifică câte agenții îndeplinesc condițiile date și în a doua etapă le rezervă.

Figura 4.5 și Figura 4.6 arată agenția Nod2 în momentul pre rezervării și în momentul rezervării.

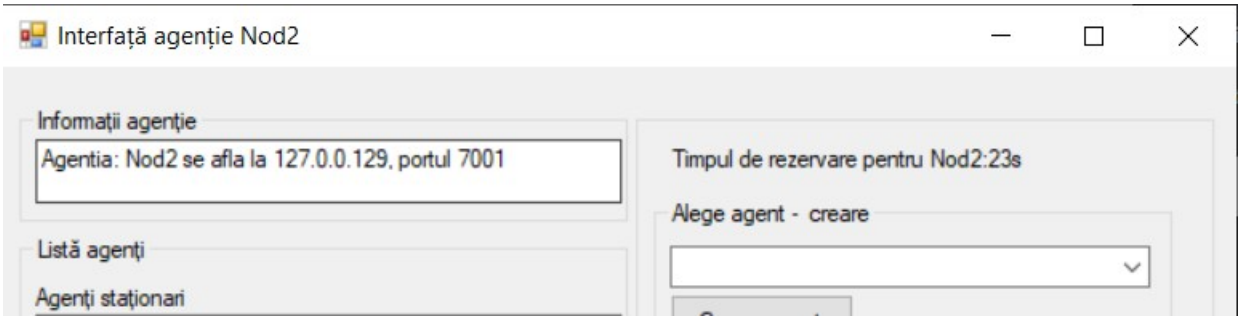


Figura 4.5: Agenție pre rezervată

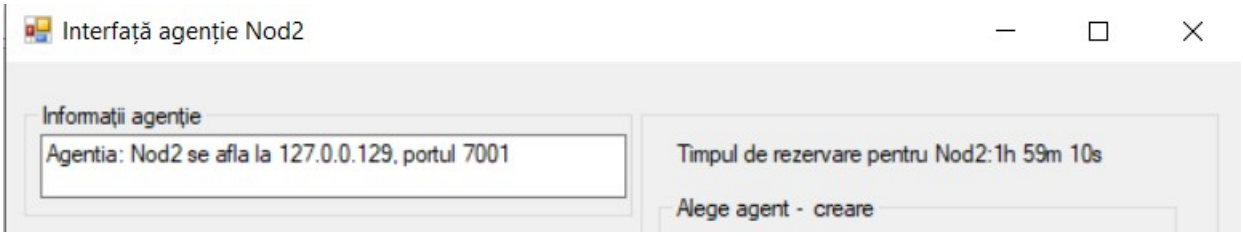


Figura 4.6: Agenție rezervată

### *4.3. Memorie, procesor și limitări*

Aplicația a fost rulată pe un laptop cu sistem de operare Windows 10, procesor 2.8 GHz. Nu necesită memorie suplimentară și nici procesor performant. Singura limitare este cea a sistemului de operare, deoarece aplicația este dezvoltată în limbajul de programare C#, și prin urmare este nevoie de sistemul de operare Windows.

Cât despre limitarea la nivel de aplicație, o problemă majoră pentru agenții care se întorc la sursă ar fi deconectarea bruscă a agenției sau a unei agenții aflate în drumul de întoarcere, deoarece agentul există atât timp cât el este găzduit în cadrul unei agenții, iar starea lui se resetează în momentul când își termină sarcina complet. Dacă traseul lui se termină brusc, agentul este pierdut în rețea. Același lucru se întâmplă și cu agentul care creează clone, deoarece el este master-ul, la el se întorc clonele cu informații, iar dacă agenția în care rulează masterul este deconectată, clonele rămân necontrolate.

# Concluzii și direcții viitoare

Încă de la început, obiectivul principal a fost proiectarea unui platforme distribuite care utilizează agenții mobili pentru efectuarea anumitor sarcini, iar agenții mobili să respecte atributele generale: autonomie, execuție în conformitate cu mediu de rulare, mobilitate în rețea, De-a lungul dezvoltării proiectul, obiectivul a rămas același însă modalitatea de abordare și implementare a suferit modificări, în funcție cerințele sistemului sau de gradul de dificultate apărut.

În comparație cu alte proiecte similare, proiectul nu prezintă un grad de dificultate ridicat, ci reprezintă o aplicație care realizează comunicarea a mai multor computere conectate la aceeași rețea de Internet.

Proiectul poate fi dezvoltat în mai multe direcții, astfel încât să se implementeze un sistem riguros, flexibil care se poate plia pe diferite aplicații cu o funcționalitate completă și care să satisfacă toate necesitățile apărute datorită tehnologiei sau a mediului de desfășurare. Direcțiile viitoare ale aplicației pot fi:

1. *Sistem general de comunicație*

O proprietate importantă a agenților este aceea de a comunica între ei. Comunicarea între agenți trebuie susținută de un sistem de comunicație puternic, în care agenții nu se pot "cunoaște" neapărat între ei, dar pot schimba mesaje. Deci, o direcție viitoare a proiectului o constituie un model de comunicare care este flexibil, extensibil, folosind structuri generale de transmitere a mesajelor. Schimbul de mesaje trebuie să suporte atât comunicarea sincronă, cat și asincronă. Acest mecanism de transmiterea a mesajelor vine și in ajutorul erorilor în cazul în care un agent este pierdut în rețea, deoarece agenția sursă a fost deconectată. De exemplu, poate primi un răspuns din partea agenției cum că fost reconectată și poate să își reia execuția.

2. *Optimizarea procesului de migrare*

Migrarea unui agent reprezintă pus și simplu procesul de transfer al agentului de la un calculator la altul. În timpul procesul de migrare, agenția expeditoare și agenția receptoare trebuie să comunice prin rețea și să facă schimb de date despre agentul care dorește să migreze. Asemeni unui protocol de comunicare, putem defini procesul cu ajutorul unui protocol de migrare. Trebuie să existe anumite reguli de respectat și strategii în procesul de migrare, pentru evitarea erorilor care pot afecta întreg sistemul. Pentru a optimiza procesul de migrare, literatura de specialitate a implementat în anumite sisteme bazate pe agenți mobil mai multe tipuri de strategii. Ca o direcție viitoare a proiectului, pot fi aplicate strategiile respective astfel încât oricât de amplu ar deveni proiectul, procesul de migrare să nu fie afectat.

3. *Securitate*

În general, dezvoltarea unei aplicații necesită definirea cerințelor, fie ele funcționale care descriu sarcinile pe care trebuie să le îndeplinească sau definirea nefuncțională modul în care sunt îndeplinite aceste sarcini. Securitatea se numără printre cerințele nefuncționale, care reprezintă o mare provocare pentru extinderea noilor tehnologii. Este adesea legată de design și identificarea cerințelor sistemului implicat, fără a compromite funcționalitatea aplicației.

Atacurile care apar într-un sistem bazat pe agenți mobili pot avea diferite surse și diferite ținte. Pentru a rezista acestor atacuri trebuie avute în vedere aspecte importante cum ar fi: autentificarea, confidențialitatea, integritatea și controlul accesului la diferite resurse. Aceste aspecte pot fi luate în considerare pentru dezvoltarea proiectului.

În concluzie, utilizarea agenților mobili a condus la desprinderea unor avantaje importante, după cum urmează:

- din punct de vedere al proiectării, abstractizarea agentului agentul mobil conduce la un bun management a aplicațiilor care sunt construite cu ajutorul lor;
- agenții mobili încapsulează comportamentul dinamic și se modifică în funcție de mediul de execuție;
- mobilitatea agenților, capacitatea de a migra in rețea, ii face candidații perfecți pentru utilizarea în mediile mobile și distribuite.

# Bibliografie

[1] Hind Idrissi, „Contributions to the Security of Mobile AgentSystems”, , , pp. 7, 2016.

[2] Peter Braun, „The Migration Processof Mobile Agents, Implementation, Classification, and Optimization”, , , pp. , 2003.

[3] Maarten van SteenAndrew, S. Tanenbaum, „Distributed Systems”, , 2018.

[4] Danny B. Lange and Mitsuru Oshima, „Programming and Deploying Mobile Agents with Java”,”, , 1998.

[5] Microsoft .Net, Serialization Concepts [Online], Disponibil la adresa: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/, Accesat: 2019.

[6] Danny B. Lange, Mitsuru Oshima, „Mobile Agents with Java: The Aglet API”, To appear in World Wide Web (Baltzer Science Publishers, The Netherlands), , pp. , 1998.

[7] Dilyana Staneva, Petya Gacheva, „Building distributed applications with Java mobile agents”, International Workshop NGNT, , pp. 103-109, .

[8] JADE, JAVA Agent DEvelopment Framework [Online], Disponibil la adresa: http://jade.tilab.com, Accesat: 2019.

[9] Richard Blum, „C# Network Programming”, , 2003.

[10] Jason J Holdsworth, „The Nature of Breath-First Search”, , , pp. 2, 1999.

# Anexe.

## *Anexa 1. Diagrama de clase a aplicației*

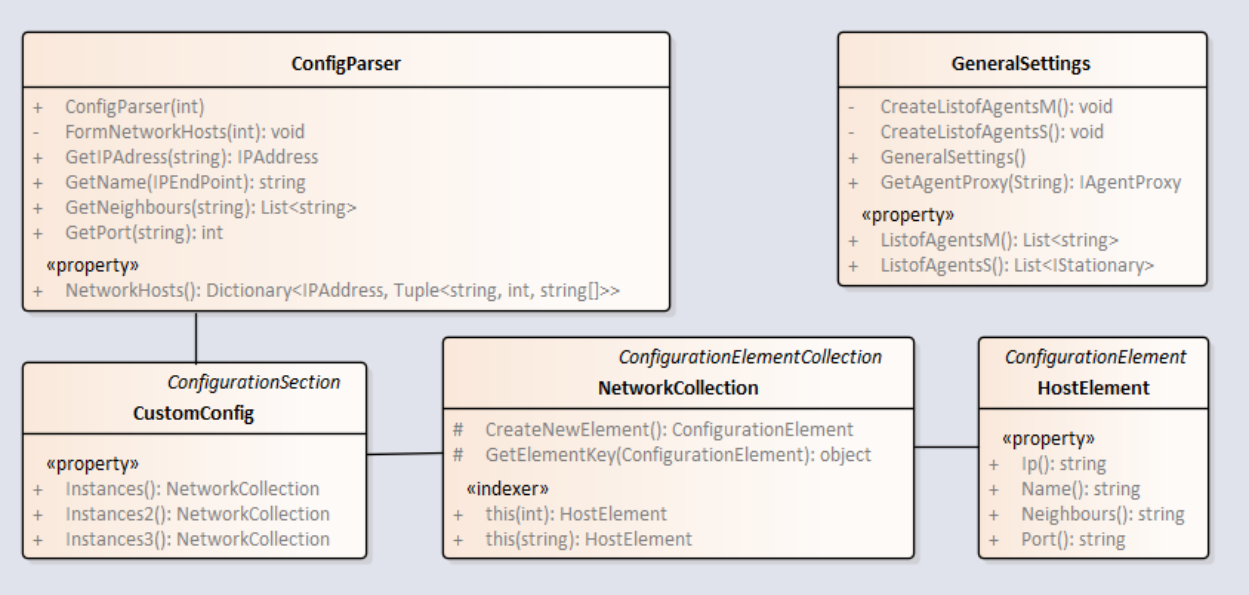


Figura 1. Clasele adiționale (parsarea fișierului de configurație și setările generale a agenților creați)

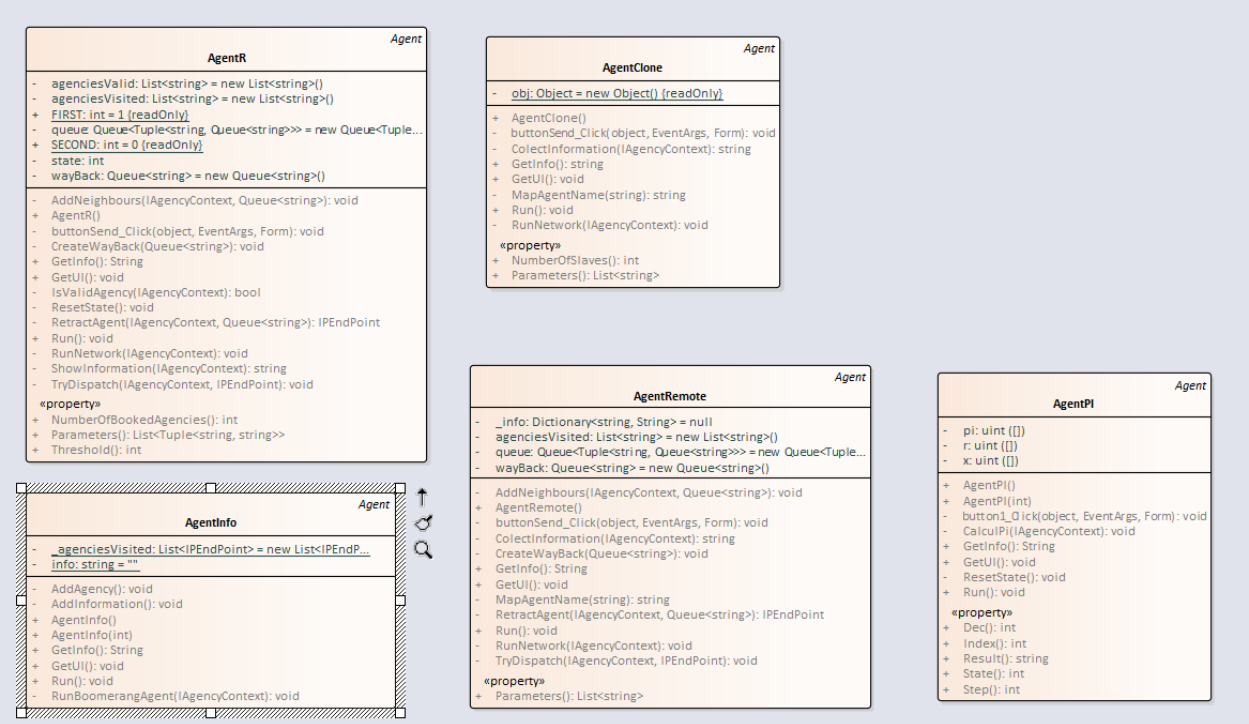


Figura 2. Clasele agenților mobili creați (derivate din clasa de bază Agent)

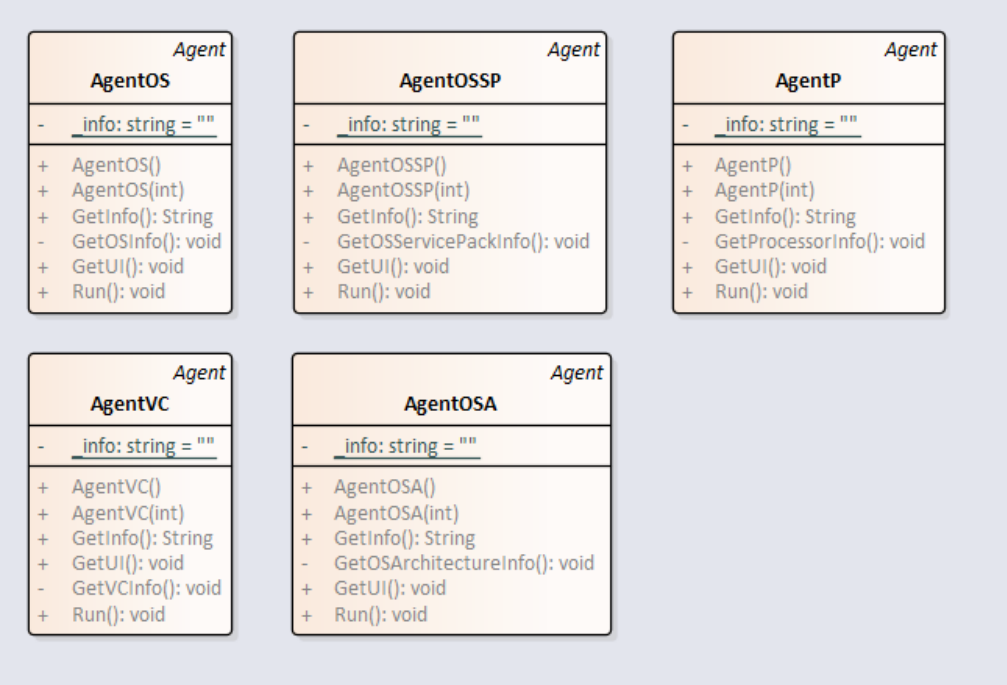


Figura 3. Clasele agenților staționari creați (derivate din clasa de baza Agent)

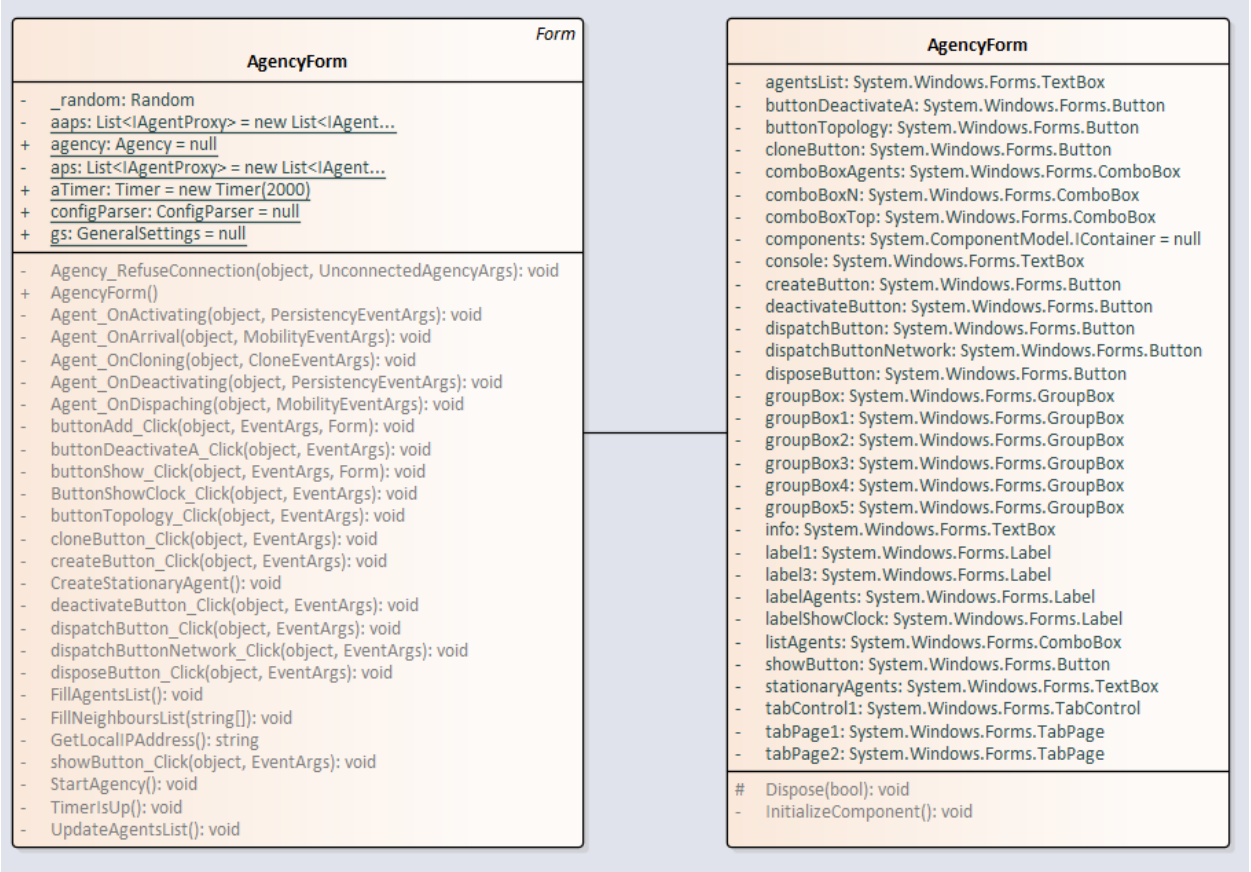


Figura 4. Clasa interfeței grafice

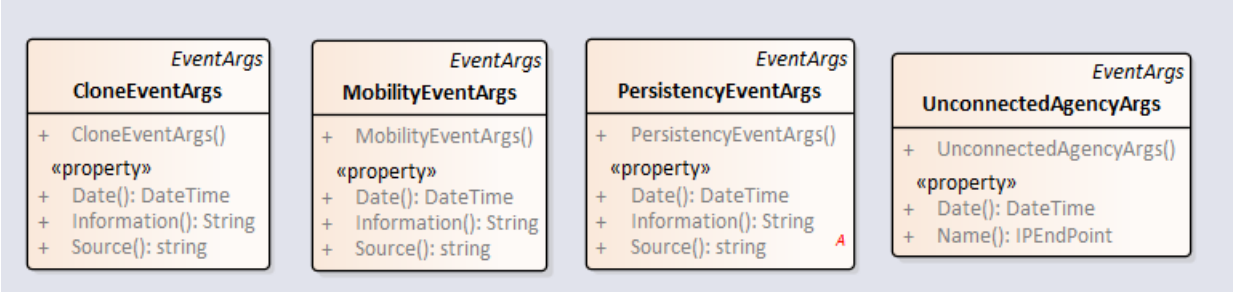


Figura 5. Clasele pentru argumente de evenimente (derivate din clasa EventArgs)

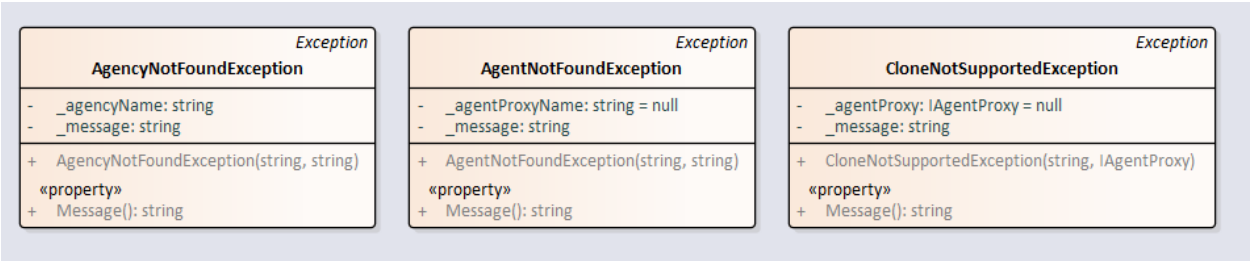


Figura 6. Clasele pentru excepții (derivate din clasa de baza Exception)

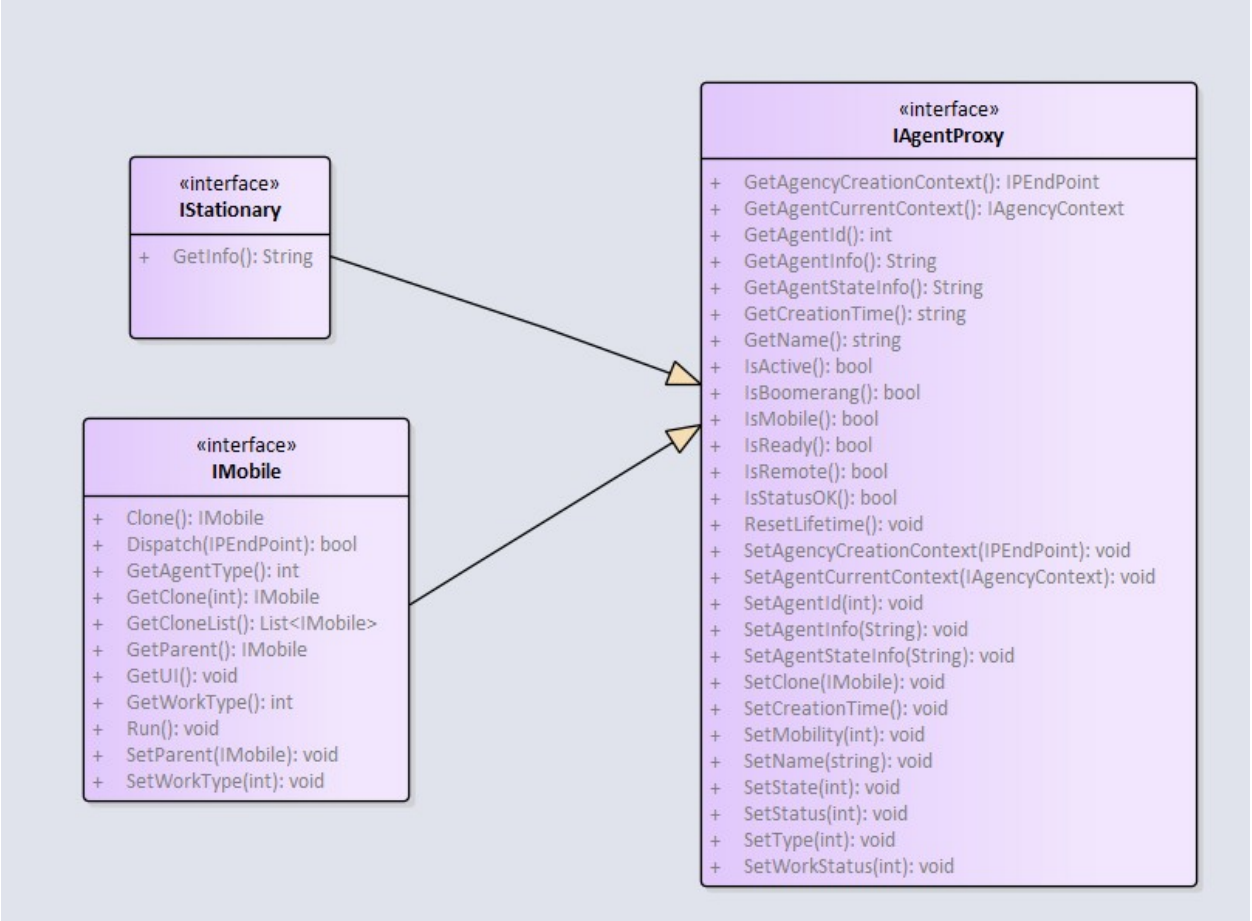


Figura 7. Interfețele folosite de către agenți

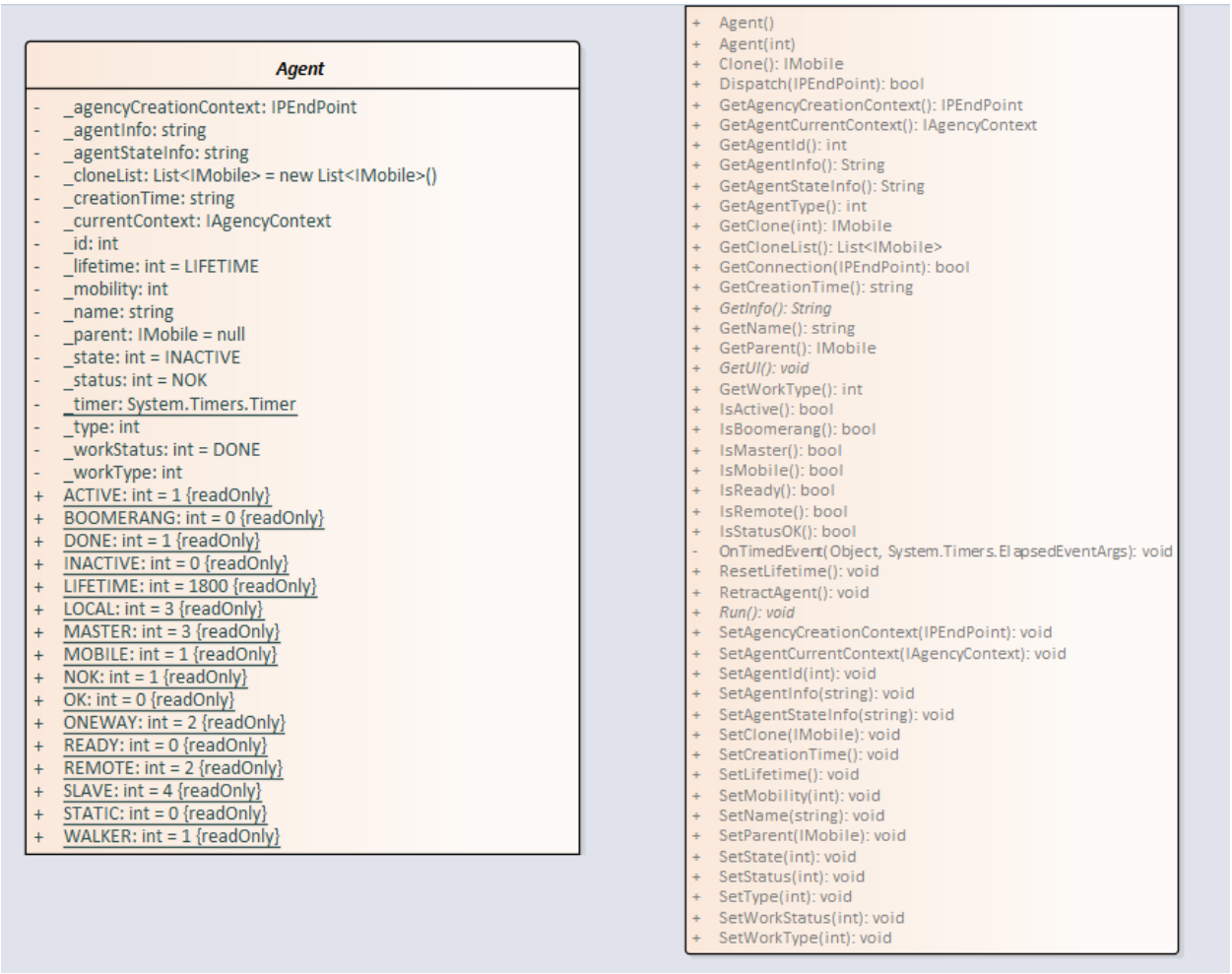


Figura 8. Clasa abstractă Agent (implementează interfețele IMobile si IStationary)

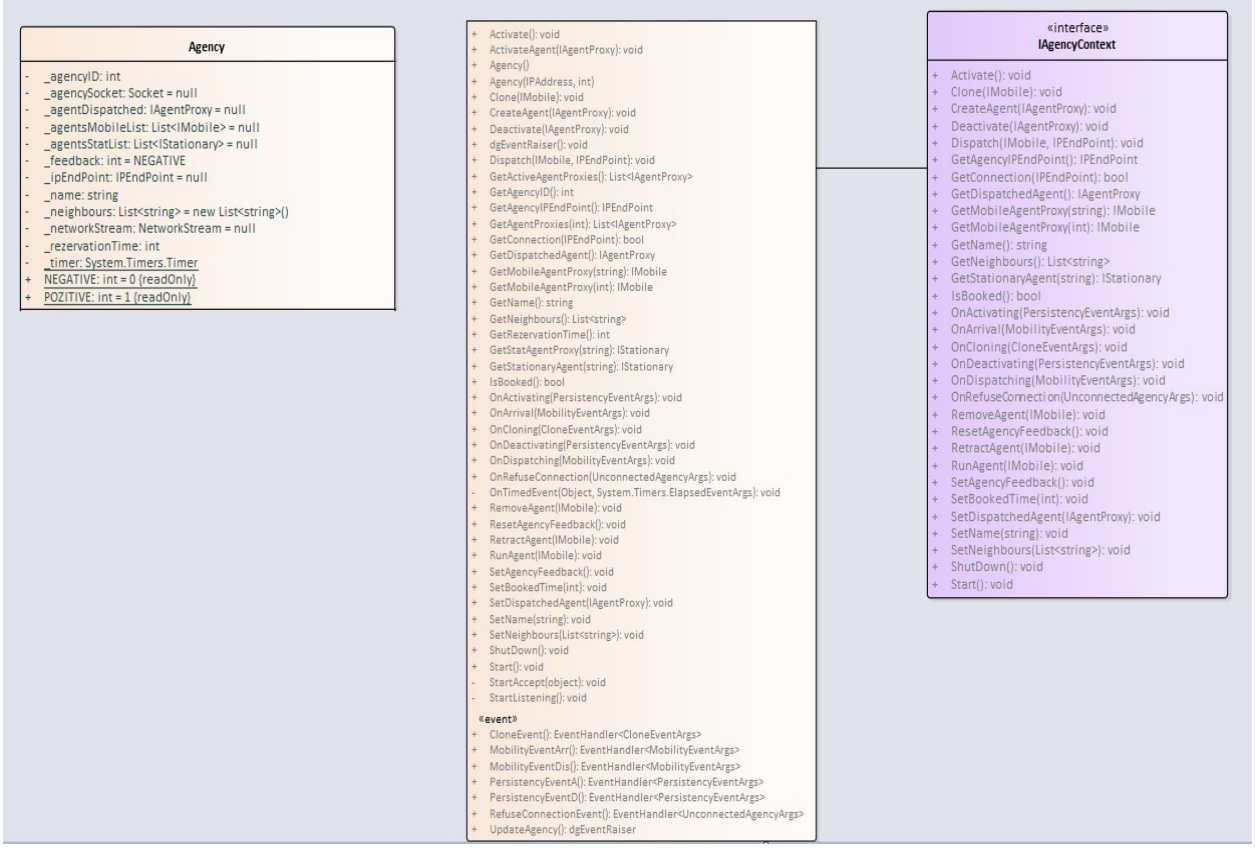


Figura 9. Clasa agenției implementând interfața IAgencyContext

## *Anexa 2. Codul sursă pentru agenții implementați*

```
//Codul complet pentru agentul care creează clone și le expediază pentru a colecta
informațiile

using MobileAgent.AgentManager;
using MobileAgent.EventAgent;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Text;
using System.Threading;
using System.Windows.Forms;

namespace AgentApp.Agents.MobileAgents
{
    [Serializable]
    public class AgentClone : Agent, IMobile
    {
        #region Private Fields
        private static readonly Object obj = new Object();
        #endregion Private Fields
```

```
        #region Constructor
        /// <summary>
        /// Constructorul care setează datele specifice ale agentului
        /// </summary>
        public AgentClone():base()
        {
            SetType(WALKER);
            SetName("AgentClone");
            SetAgentInfo("Colecteaza informatii din retea.");
        }
        #endregion Constructor

        /// <summary>
        /// Datele de stare ale agentului
        /// </summary>
        #region Properties
        public List<string> Parameters { get; set; } = new List<string>();
        public int NumberOfSlaves { get; set; } = 0;
        #endregion Properties

        #region Private Methods
        /// <summary>
        /// Metoda care lanseaza clone in retea si aduna informatiile
        /// </summary>
        /// <param name="agencyContext">contextul agentiei</param>
        private void RunNetwork(IAgencyContext agencyContext)
        {
            MobilityEventArgs args = new MobilityEventArgs();
            CloneEventArgs argsc = new CloneEventArgs();
            if
(agencyContext.GetAgencyIPEndPoint().Equals(GetAgencyCreationContext()))
            {
                if (IsMaster())
                {
                    SetAgentStateInfo("");
                    foreach (String n in agencyContext.GetNeighbours())
                    {
                        IPAddress ipAddress =
AgencyForm.configParser.GetIPAdress(n);
                        int portNumber = AgencyForm.configParser.GetPort(n);
                        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress,
portNumber);
                        if (agencyContext.GetConnection(ipEndPoint))
                        {
                            IMobile agentCloned = this.Clone();


                            agentCloned.SetParent(this);
                            agentCloned.SetWorkType(SLAVE);
                            NumberOfSlaves++;
```

```
                            agentCloned.SetAgentId(GetAgentId() + NumberOfSlaves);
                            agentCloned.SetName(GetName() + "_cloned_" +
NumberOfSlaves);
                            agentCloned.GetParent().SetAgentCurrentContext(null);

                            argsc.Source = "Agentul " + GetName();
                            argsc.Information = "A creat clona : " +
agentCloned.GetName();
                            agencyContext.OnCloning(argsc);

                            agencyContext.Dispatch(agentCloned, ipEndPoint);
                        }
                    }
                    while (GetCloneList().Count() < NumberOfSlaves) ;

                    if (!GetAgentStateInfo().Equals(""))
                    {
                        args.Source = "Punct de stop: ";
                        args.Information = "Agentul  " + GetName() + " a adunat
informațiile: " + Environment.NewLine + GetAgentStateInfo();
                        agencyContext.OnArrival(args);
                        Console.Beep(800, 1000);
                    }
                    else
                    {
                        args.Source = "Punct de stop: ";
                        args.Information = "Agentul " + GetName() + " nu a adunat
nicio informație !";
                        agencyContext.OnArrival(args);
                        Console.Beep(800, 1000);
                    }
                    GetCloneList().Clear();
                }
                else
                {
                    IMobile parent =
agencyContext.GetMobileAgentProxy(GetParent().GetAgentId());
                    lock (obj)
                    {
                        parent.SetAgentStateInfo(parent.GetAgentStateInfo() +
GetAgentStateInfo());
                        parent.SetClone(this);
                    }
                    agencyContext.RemoveAgent(this);

                }
            }
            else
            {
                if (!agencyContext.IsBooked())
                {
```

```
                    args.Source = "Punct de rulare: ";
                    args.Information = "Agentul  " + GetName() + " rulează..." +
Environment.NewLine + agencyContext.GetName() + ": " +
ColectInformation(agencyContext);
                    agencyContext.OnArrival(args);
                    Console.Beep();
                    agencyContext.Dispatch(this, GetAgencyCreationContext());
                }
                else
                {
                    args.Source = "Punct de rulare: ";
                    args.Information = "Agentul  " + GetName() + " nu poate rula!
Agentie rezervată";
                    agencyContext.OnArrival(args);
                    Console.Beep();
                    agencyContext.Dispatch(this, GetAgencyCreationContext());

                }
            }

        }
        /// <summary>
        /// Metoda care colecteaza informatiile de la agentii stationari
        /// </summary>
        /// <param name="agencyContext">contextul agentiei</param>
        /// <returns></returns>
        private string ColectInformation(IAgencyContext agencyContext)
        {
            string information = "";
            foreach (string par in Parameters)
            {
                IStationary agentStatic =
agencyContext.GetStationaryAgent(MapAgentName(par));
                String i = agentStatic.GetInfo();
                information += i;

            }
            SetAgentStateInfo(GetAgentStateInfo() + agencyContext.GetName() + ": "
+ information + Environment.NewLine);
            return information;
        }
        /// <summary>
        /// Metoda care face conversia intre parametrii de intrare si agentul
stationar necesar
        /// </summary>
        /// <param name="parameter">resursa indicata</param>
        /// <returns>agentul stationar care detine resursa</returns>
        private string MapAgentName(string parameter)
        {
            string type = "";
            switch (parameter)
```

```
            {
                case "Sistem de operare":
                    {
                        type = "AgentOS";
                        break;
                    }
                case "Arhitectură sistem de operare":
                    {
                        type = "AgentOSA";
                        break;
                    }
                case "Service Pack sistem de operare":
                    {
                        type = "AgentOSSP";
                        break;
                    }
                case "Informații procesor":
                    {
                        type = "AgentP";
                        break;
                    }
                case "Informații placă video":
                    {
                        type = "AgentVC";
                        break;
                    }
                default:
                    {
                        break;
                    }
            }
            return type;
        }
        /// <summary>
        /// Controlerul care seteaza parametrii agentului din interfata grafica
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        /// <param name="ui"></param>
        private void buttonSend_Click(object sender, EventArgs e, Form ui)
        {
            try
            {
                List<string> _parameters = new List<string>();
                foreach (Control c in ui.Controls)
                {
                    if (c is CheckedListBox)
                    {
                        CheckedListBox control = (CheckedListBox)c;
                        foreach (object itemchecked in control.CheckedItems)
                        {
```

```
                            _parameters.Add(itemchecked.ToString());
                        }
                        break;
                    }

                }
                Parameters = _parameters;
                ui.Close();
            }
            catch (NullReferenceException nre)
            {
                MessageBox.Show("NullReferenceException !" + nre.Message + " -->
Trimite parametrii agentului!");

            }
            catch (Exception ex)
            {
                MessageBox.Show("Exception !" + ex.Message + " --> Trimite
parametrii agentului!");
            }
        }
        #endregion Private Methods

        #region Public Override Methods
        /// <summary>
        /// Metoda Run() specifica agentului
        /// </summary>
        public override void Run()
        {
            IAgencyContext agencyContext = GetAgentCurrentContext();
            RunNetwork(agencyContext);
        }
        /// <summary>
        /// Metoda de afisare a interfetei grafice specifica agentului
        /// </summary>
        public override void GetUI()
        {
            Form ui = new Form();

            Label label1;
            Button button1;
            CheckedListBox checkedListBox1;

            label1 = new Label();
            button1 = new Button();
            checkedListBox1 = new CheckedListBox();
            ui.SuspendLayout();
            //
            // label1
            //
            label1.AutoSize = true;
```

```
            label1.Location = new System.Drawing.Point(13, 25);
            label1.Name = "label1";
            label1.Size = new System.Drawing.Size(65, 17);
            label1.TabIndex = 5;
            label1.Text = "Informatii";
            //
            // button1
            //
            button1.Location = new System.Drawing.Point(301, 89);
            button1.Margin = new Padding(3, 2, 3, 2);
            button1.Name = "button1";
            button1.Size = new System.Drawing.Size(107, 38);
            button1.TabIndex = 4;
            button1.Text = "Trimite";
            button1.UseVisualStyleBackColor = true;

            //
            // checkedListBox1
            //
            checkedListBox1.CheckOnClick = true;
            checkedListBox1.FormattingEnabled = true;
            checkedListBox1.Items.AddRange(new object[] {
            "Sistem de operare",
            "Arhitectură sistem de operare",
            "Service Pack sistem de operare",
            "Informații procesor",
            "Informații placă video"});
            checkedListBox1.Location = new System.Drawing.Point(13, 58);
            checkedListBox1.Margin = new Padding(4);
            checkedListBox1.Name = "checkedListBox1";
            checkedListBox1.Size = new System.Drawing.Size(259, 106);
            checkedListBox1.TabIndex = 6;
            //
            // AgentRemoteUI
            //
            ui.AutoScaleDimensions = new System.Drawing.SizeF(8F, 16F);
            ui.AutoScaleMode = AutoScaleMode.Font;
            ui.ClientSize = new System.Drawing.Size(451, 177);
            ui.Controls.Add(checkedListBox1);
            ui.Controls.Add(label1);
            ui.Controls.Add(button1);
            ui.Margin = new Padding(3, 2, 3, 2);
            ui.Name = "AgentRemoteUI";
            ui.Text = "Interfață AgentRemote";
            ui.ResumeLayout(false);
            ui.PerformLayout();

            button1.Click += new System.EventHandler((sender, e) =>
buttonSend_Click(sender, e, ui));
            if (ui.Controls.Count != 0)
            {
```

```
                var thread = new Thread(() =>
                {
                    Application.Run(ui);
                });
                thread.Start();
            }
        }
        public override string GetInfo()
        {
            throw new NotImplementedException();
        }
        #endregion Public Override Methods
    }
}

//Codul sursă complet pentru agentul care parcurge rețeaua pentru a colecta
informațiile  date ca parametrii

using MobileAgent.AgentManager;
using MobileAgent.EventAgent;
using System;
using System.Collections.Generic;
using System.Net;
using System.Threading;
using System.Windows.Forms;


namespace AgentApp.Agents
{
    [Serializable]
    public class AgentRemote : Agent, IMobile
    {
        #region Private Fields
        /// <summary>
        /// Variabile de stare pentru agent
        /// </summary>
        Queue<string> wayBack = new Queue<string>();
        Dictionary<string, String> _info = null;
        List<string> agenciesVisited = new List<string>();
        Queue<Tuple<string, Queue<string>>> queue = new Queue<Tuple<string,
Queue<string>>>();
        #endregion Private Fields

        #region Constructors
        /// <summary>
        /// Constructorul care setează datele specifice ale agentului
        /// </summary>
        public AgentRemote() : base()
        {
            Parameters = new List<string>();
            SetType(Agent.WALKER);
```

```
            SetName("AgentRemote");
            SetAgentInfo("Colectează informații din rețea.");
            _info = new Dictionary<string, String>();
        }
        #endregion Constructors

        #region Properties
        public List<string> Parameters { get; set; }
        #endregion Properties

        #region Private Methods
        /// <summary>
        /// Metoda care face conversia intre datele interfetei grafice si numele
agentului stationar
        /// </summary>
        /// <param name="parameter">resursa ceruta</param>
        /// <returns>numele agentului stationar care detine resursa</returns>
        private string MapAgentName(string parameter)
        {
            string type = "";
            switch (parameter)
            {
                case "Sistem de operare":
                    {
                        type = "AgentOS";
                        break;
                    }
                case "Arhitectură sistem de operare":
                    {
                        type = "AgentOSA";
                        break;
                    }
                case "Service Pack sistem de operare":
                    {
                        type = "AgentOSSP";
                        break;
                    }
                case "Informații procesor":
                    {
                        type = "AgentP";
                        break;
                    }
                case "Informații placă video":
                    {
                        type = "AgentVC";
                        break;
                    }
                default:
                    {
                        break;
                    }
```

```
            }
            return type;
        }
        /// <summary>
        /// Metoda care adaugă vecinii nevizitati in coada
        /// </summary>
        /// <param name="agencyContext">contextul agentiei</param>
        /// <param name="agencyQueue">coada care contine nodurile agentiei</param>
        private void AddNeighbours(IAgencyContext agencyContext, Queue<string>
agencyQueue)
        {
            foreach (string n in agencyContext.GetNeighbours())
            {
                if (!agenciesVisited.Contains(n))
                {
                    Queue<string> q = new Queue<string>();
                    q.Enqueue(agencyContext.GetName());
                    foreach (Object obj in agencyQueue)
                    {
                        q.Enqueue(obj.ToString());
                    }
                    queue.Enqueue(Tuple.Create(n, q));
                    agenciesVisited.Add(n);
                }
            }
        }
        /// <summary>
        /// Metoda de expediere a agentului
        /// </summary>
        /// <param name="agencyContext">contextul agentiei</param>
        /// <param name="destination">punctul destinatie</param>
        private void TryDispatch(IAgencyContext agencyContext, IPEndPoint
destination)
        {
            MobilityEventArgs args = new MobilityEventArgs();
            if (!agencyContext.GetConnection(destination))
            {

                if (queue.Count != 0)
                {
                    Tuple<string, Queue<string>> t = queue.Dequeue();
                    if (queue.Count != 0)
                    {
                        string next = queue.Peek().Item1;
                        if (agencyContext.GetNeighbours().Contains(next))
                        {
                            IPAddress ipAddress =
AgencyForm.configParser.GetIPAdress(next);
                            int portNumber =
AgencyForm.configParser.GetPort(next);
                            destination = new IPEndPoint(ipAddress, portNumber);
```

```
                            args.Source = "Punct de plecare: ";
                            args.Information = "Agentul " + GetName() + " se duce
către " + next;
                            agencyContext.OnDispatching(args);

                        }
                        else
                        {
                            CreateWayBack(t.Item2);
                            wayBack.Dequeue();
                            string back = wayBack.Peek();
                            IPAddress ipAddress =
AgencyForm.configParser.GetIPAdress(back);
                            int portNumber =
AgencyForm.configParser.GetPort(back);
                            destination = new IPEndPoint(ipAddress, portNumber);

                            args.Source = "Punct de plecare: ";
                            args.Information = "Agentul " + GetName() + " se duce
către " + back;
                            agencyContext.OnDispatching(args);
                        }
                    }
                    else
                    {
                        t.Item2.Dequeue();
                        destination = RetractAgent(agencyContext, t.Item2);
                        if (destination == null)
                        {
                            args.Source = "Punct de stop: ";
                            args.Information = "Agentul " + GetName() + " nu a
adunat nicio informație !";
                            agencyContext.OnArrival(args);
                            Console.Beep(800, 1000);

                            return;
                        }

                        args.Source = "Punct de plecare: ";
                        args.Information = "Agentul " + GetName() + " se duce
către sursă";
                        agencyContext.OnDispatching(args);
                    }

                }
                else
                {
                    return;

                }
```

```
                TryDispatch(agencyContext, destination);
            }
            else
            {
                agencyContext.Dispatch(this, destination);
            }
        }
        /// <summary>
        /// Metoda care creaza drumul de intorcere in retea
        /// </summary>
        /// <param name="b">coada care contine nodurile drumului de
intoarcere</param>
        private void CreateWayBack(Queue<string> b)
        {
            wayBack = b;
            List<string> q = new List<string>(queue.Peek().Item2.ToArray());
            q.Reverse();
            q.RemoveAt(0);
            foreach (string el in q)
            {
                wayBack.Enqueue(el);
            }
        }
        /// <summary>
        /// Metoda care creeaza drumul de intorcere la sursa
        /// </summary>
        /// <param name="agencyContext">contextul agentiei</param>
        /// <param name="t">coada care contine drumul de intrcere la sursa</param>
        /// <returns>urmatorul punct destinatar</returns>
        private IPEndPoint RetractAgent(IAgencyContext agencyContext,
Queue<string> t)
        {
            IPEndPoint destination = null;
            SetWorkStatus(Agent.DONE);
            wayBack = t;
            if (wayBack.Count != 0)
            {
                string back = wayBack.Dequeue();
                IPAddress ipAddress = AgencyForm.configParser.GetIPAdress(back);
                int portNumber = AgencyForm.configParser.GetPort(back);
                destination = new IPEndPoint(ipAddress, portNumber);
            }
            else
            {
                if (!
agencyContext.GetAgencyIPEndPoint().Equals(GetAgencyCreationContext()))
                {
                    destination = GetAgencyCreationContext();
                }
```

```
            }
            return destination;
        }
        /// <summary>
        /// Metoda care parcurge intreaga topologie de retea
        /// </summary>
        /// <param name="agencyContext">contextul agentiei</param>
        private void RunNetwork(IAgencyContext agencyContext)
        {
            MobilityEventArgs args = new MobilityEventArgs();
            if (wayBack.Count != 0)
            {

                wayBack.Dequeue();
                if (queue.Count != 0)
                {
                    args.Source = "Punct de sosire: ";
                    args.Information = "Agentul " + GetName() + " se întoarce
pentru a se duce la " + queue.Peek().Item1;
                    agencyContext.OnArrival(args);
                }
                else
                {
                    args.Source = "Punct de sosire: ";
                    args.Information = "Agentul " + GetName() + " se întoarce la
sursa";
                    agencyContext.OnArrival(args);
                }
                if (wayBack.Count != 0)
                {
                    string back = wayBack.Peek();
                    IPAddress ipAddress =
AgencyForm.configParser.GetIPAdress(back);
                    int portNumber = AgencyForm.configParser.GetPort(back);
                    IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, portNumber);

                    args.Source = "Punct de plecare: ";
                    args.Information = "Agentul " + GetName() + " se duce către "
+ back;
                    agencyContext.OnDispatching(args);

                    TryDispatch(agencyContext, ipEndPoint);
                }
                else
                {
                    if (queue.Count != 0)
                    {
                        string cont = queue.Peek().Item1;
                        IPAddress ipAddress =
AgencyForm.configParser.GetIPAdress(cont);
```

```
                        int portNumber = AgencyForm.configParser.GetPort(cont);
                        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress,
portNumber);

                        args.Source = "Punct de plecare: ";
                        args.Information = "Agentul " + GetName() + " se duce
către " + cont;
                        agencyContext.OnDispatching(args);

                        TryDispatch(agencyContext, ipEndPoint);
                    }
                    else
                    {
                        args.Source = "Punct de plecare: ";
                        args.Information = "Agentul " + GetName() + " se duce
către sursă";
                        agencyContext.OnDispatching(args);

                        TryDispatch(agencyContext, GetAgencyCreationContext());
                    }
                }
            }
            else
            {

                if
(agencyContext.GetAgencyIPEndPoint().Equals(GetAgencyCreationContext()))
                {
                    if (IsReady())
                    {
                        agenciesVisited.Clear();
                        _info.Clear();
                        SetAgentStateInfo("");
                        agenciesVisited.Add(agencyContext.GetName());
                        AddNeighbours(agencyContext, new Queue<string>());
                        if (queue.Count != 0)
                        {
                            string next = queue.Peek().Item1;
                            IPAddress ipAddress =
AgencyForm.configParser.GetIPAdress(next);
                            int portNumber =
AgencyForm.configParser.GetPort(next);
                            IPEndPoint ipEndPoint = new IPEndPoint(ipAddress,
portNumber);

                            args.Source = "Punct de plecare: ";
                            args.Information = "Agentul " + GetName() + " se duce
către " + next;
                            agencyContext.OnDispatching(args);

                            TryDispatch(agencyContext, ipEndPoint);
```

```
                        }

                    }
                    else
                    {
                        if (!GetAgentStateInfo().Equals(""))
                        {
                            args.Source = "Punct de stop: ";
                            args.Information = "Agentul  " + GetName() + " a
adunat informațiile: " + Environment.NewLine + GetAgentStateInfo();
                            agencyContext.OnArrival(args);
                            Console.Beep(800, 1000);
                        }
                        else
                        {
                            args.Source = "Punct de stop: ";
                            args.Information = "Agentul " + GetName() + " nu a
adunat nicio informație !";
                            agencyContext.OnArrival(args);
                            Console.Beep(800, 1000);
                        }

                    }
                }
                else
                {
                    if (queue.Count != 0)
                    {
                        Tuple<string, Queue<string>> t = queue.Dequeue();
                        if (!agencyContext.IsBooked())
                        {
                            args.Source = "Punct de rulare: ";
                            args.Information = "Agentul  " + GetName() + "
rulează..." + Environment.NewLine + agencyContext.GetName() + ": " +
ColectInformation(agencyContext);
                            agencyContext.OnArrival(args);
                            Console.Beep();
                        }
                        else
                        {
                            args.Source = "Punct de rulare: ";
                            args.Information = "Agentul  " + GetName() + " nu
poate rula: Agenție rezervată";
                            agencyContext.OnArrival(args);
                            Console.Beep();
                        }

                        AddNeighbours(agencyContext, t.Item2);
                        if (queue.Count != 0)
                        {
```

```
                            string next = queue.Peek().Item1;
                            if (agencyContext.GetNeighbours().Contains(next))
                            {
                                IPAddress ipAddress =
AgencyForm.configParser.GetIPAdress(next);
                                int portNumber =
AgencyForm.configParser.GetPort(next);
                                IPEndPoint ipEndPoint = new IPEndPoint(ipAddress,
portNumber);

                                args.Source = "Punct de plecare: ";
                                args.Information = "Agentul " + GetName() + " se
duce către " + next;
                                agencyContext.OnDispatching(args);

                                TryDispatch(agencyContext, ipEndPoint);

                            }
                            else
                            {
                                CreateWayBack(t.Item2);
                                string back = wayBack.Peek();
                                IPAddress ipAddress =
AgencyForm.configParser.GetIPAdress(back);
                                int portNumber =
AgencyForm.configParser.GetPort(back);
                                IPEndPoint ipEndPoint = new IPEndPoint(ipAddress,
portNumber);

                                args.Source = "Punct de plecare: ";
                                args.Information = "Agentul " + GetName() + " se
întoarce la " + back;
                                agencyContext.OnDispatching(args);

                                TryDispatch(agencyContext, ipEndPoint);
                            }
                        }
                        else
                        {
                            SetWorkStatus(Agent.DONE);

                            args.Source = "Punct de plecare: ";
                            args.Information = "Agentul " + GetName() + " se
întoarce la sursă.";
                            agencyContext.OnDispatching(args);

                            TryDispatch(agencyContext,
GetAgencyCreationContext());
                        }
                    }
                }
```

```
            }
        }
        /// <summary>
        /// Metoda care colecteaza informatiile de la agentii stationari
        /// </summary>
        /// <param name="agencyContext">contextul agentiei</param>
        /// <returns></returns>
        private string ColectInformation(IAgencyContext agencyContext)
        {
            string information = "";
            foreach (string par in Parameters)
            {
                IStationary agentStatic =
agencyContext.GetStationaryAgent(MapAgentName(par));
                String i = agentStatic.GetInfo();
                if (!_info.ContainsKey(agencyContext.GetName()))
                {
                    _info.Add(agencyContext.GetName(), i);
                }
                else
                {
                    _info[agencyContext.GetName()] += i;
                }
                information += i;

            }
            SetAgentStateInfo(GetAgentStateInfo() + agencyContext.GetName() + ": "
+ _info[agencyContext.GetName()] + Environment.NewLine);
            return information;
        }
        /// <summary>
        /// Controlorul care seteaza parametrii agentului
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        /// <param name="ui"></param>
        private void buttonSend_Click(object sender, EventArgs e, Form ui)
        {
            try
            {
                List<string> _parameters = new List<string>();
                foreach (Control c in ui.Controls)
                {
                    if (c is CheckedListBox)
                    {
                        CheckedListBox control = (CheckedListBox)c;
                        foreach (object itemchecked in control.CheckedItems)
                        {
                            _parameters.Add(itemchecked.ToString());
                        }
                        break;
```

```
                    }

                }
                Parameters = _parameters;
                ui.Close();
            }
            catch (NullReferenceException nre)
            {
                MessageBox.Show("NullReferenceException !" + nre.Message + " -->
Trimite parametrii agentului!");

            }
            catch (Exception ex)
            {
                MessageBox.Show("Exception !" + ex.Message + " --> Trimite
parametrii agentului!");
            }
        }
        #endregion Private Methods

        #region Public Override Methods
        /// <summary>
        /// Metoda Run() specifica agentului
        /// </summary>
        public override void Run()
        {
            ResetLifetime();
            IAgencyContext agencyContext = GetAgentCurrentContext();
            RunNetwork(agencyContext);
        }
        /// <summary>
        /// Metoda de afisare a interfetei grafice proprii a agentului
        /// </summary>
        public override void GetUI()
        {
            Form ui = new Form();

            Label label1;
            Button button1;
            CheckedListBox checkedListBox1;

            label1 = new Label();
            button1 = new Button();
            checkedListBox1 = new CheckedListBox();
            ui.SuspendLayout();
            //
            // label1
            //
            label1.AutoSize = true;
            label1.Location = new System.Drawing.Point(13, 25);
            label1.Name = "label1";
```

```
            label1.Size = new System.Drawing.Size(65, 17);
            label1.TabIndex = 5;
            label1.Text = "Informații";
            //
            // button1
            //
            button1.Location = new System.Drawing.Point(301, 89);
            button1.Margin = new Padding(3, 2, 3, 2);
            button1.Name = "button1";
            button1.Size = new System.Drawing.Size(107, 38);
            button1.TabIndex = 4;
            button1.Text = "Trimite";
            button1.UseVisualStyleBackColor = true;

            //
            // checkedListBox1
            //
            checkedListBox1.CheckOnClick = true;
            checkedListBox1.FormattingEnabled = true;
            checkedListBox1.Items.AddRange(new object[] {
            "Sistem de operare",
            "Arhitectură sistem de operare",
            "Service Pack sistem de operare",
            "Informații procesor",
            "Informații placă video"});
            checkedListBox1.Location = new System.Drawing.Point(13, 58);
            checkedListBox1.Margin = new Padding(4);
            checkedListBox1.Name = "checkedListBox1";
            checkedListBox1.Size = new System.Drawing.Size(259, 106);
            checkedListBox1.TabIndex = 6;
            //
            // AgentRemoteUI
            //
            ui.AutoScaleDimensions = new System.Drawing.SizeF(8F, 16F);
            ui.AutoScaleMode = AutoScaleMode.Font;
            ui.ClientSize = new System.Drawing.Size(451, 177);
            ui.Controls.Add(checkedListBox1);
            ui.Controls.Add(label1);
            ui.Controls.Add(button1);
            ui.Margin = new Padding(3, 2, 3, 2);
            ui.Name = "AgentRemoteUI";
            ui.Text = "Interfață AgentRemote";
            ui.ResumeLayout(false);
            ui.PerformLayout();

            button1.Click += new EventHandler((sender, e) =>
buttonSend_Click(sender, e, ui));
            if (ui.Controls.Count != 0)
            {
                var thread = new Thread(() =>
                {
```

```
                    Application.Run(ui);
                });
                thread.Start();
            }
        }
        public override String GetInfo()
        {
            throw new NotImplementedException();
        }
        #endregion Public Override Methods

    }

}
```