Solutie de Automatizare dirijata de Produs Inteligent: Servicii de executie heterarhica a produselor în fabricația discretă, repetitiva

1. Implementarea soluţiei de conducere heterarhică a producției

1.1. Dezvoltarea conceptului de produs inteligent in vederea implementarii

Structura soluţiei de conducere heterarhică a producției este formată din următorii agenţi: Produs, Ordin şi Resursă. Agentul de tip Ordin conţine informaţii despre starea reală a produsului fizic în timpul procesului de fabricaţie.

Studiul efectuat conduce la ideea ca produsele inteligente pot fi clasificate în mai multe categorii în funcţie de capacitatea decizională şi localizarea inteligenţei. Produsul inteligent folosit în cadrul solutiei propuse are atât capacitate de stocare a informaţiilor legate de starea sa, cât şi capacitatea de a lua decizii privind modul de fabricaţie. Astfel, conform caracteristicilor prezentate în (Wong et al., 2002), produsul inteligent are o inteligenţă de nivelul 2, iar inteligenţa

este încorporată în produs.

Fig. 1.1. – Componentele produsului inteligent

Produsul inteligent folosit pentru implementarea sistemului heterarhic de conducere este alcătuit din următoarele componente (Fig. 1.1):

- Produsul fizic care va fi fabricat
- Modulul decizional – Computer-on-module (COM) Overo Air
- Port-paleta sau modulul de transport pe care sunt pozitionate produsul fizic si modulul decizional Overo Air

Produsul fizic va fi asamblat în celula de fabricaţie generica cu proces job-shop rezultând la sfârşitul unei secvente de operatii de lucru executate pe diferite resurse.

Modulul de transport sau port-paleta este dispozitivul care transportă produsul fizic şi modulul decizional în celula de fabricaţie. Port-paleta este dotată cu o eticheta RFID care poate fi scrisă şi citită. În momentul intrării port-paletei în sistem, etichetei RFID îi este atribuit un

identificator unic utilizat pentru identificarea paletelor şi transportarea acestora la posturile de lucru.

Modulul decizional este reprezentat de procesorul Overo Air (https://www.gumstix.com). Dispozitivele de tip Computer-on-module sunt dispozitive integrate care conţin: microprocesor, memorie şi porturi de intrare/ieşire.

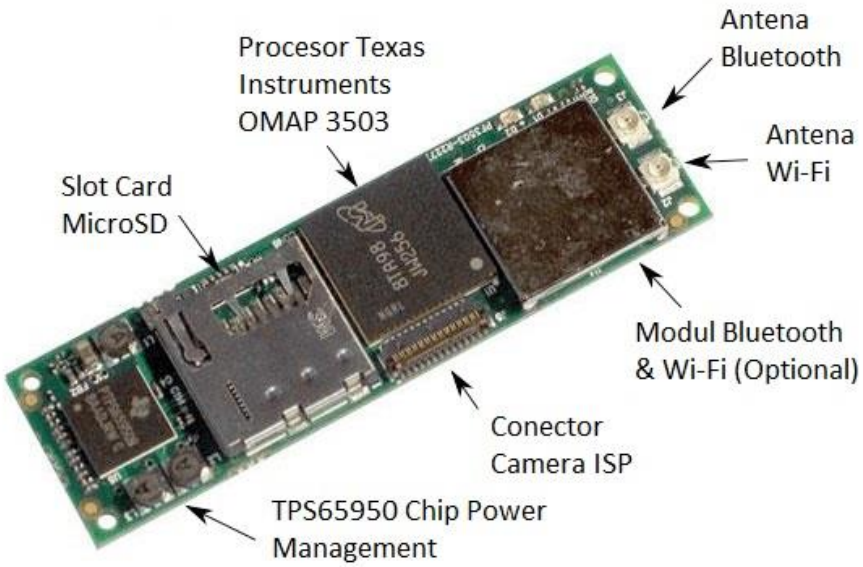
Arhitectura hardware a modulului Overo Air este reprezentată în Fig. 5.2. si este compusa din:

- Procesor Texas Instrument OMAP3503 la 600 MHz cu arhitectura ARM Cortex-A8
- Memorie RAM DDR 512MB
- Slot pentru carduri de memorie: in cadrul proiectului sunt folosite carduri de memorie microSD 16GB
- Comunicatie: Bluetooth si 802.11 wireless

Specificaţii fizice:

- Dimensiune: 58mm x 17mm x 4.2.mm
- Greutate: 5.6g
- Temperatura de functionare: 0ºC - 75ºC

Procesoarele TI OMAP3503 au la baza arhitectura OMAP 3, dedicata aplicatiilor de procesare a fisierelor de tip video si imagine dezvoltate pentru dispozitive portabile si mobile. Procesoarele OMAP (Open Multimedia Applications Platform) au, de regula, un procesor



principal de uz general din arhitectura ARM si unul sau mai multe co-procesoare dedicate.

Fig. 1.2. Prezentarea generală a componentelor Overo Air

Din punct de vedere al consumului de energie, Overo Air este alimentat prin baterii care oferă modulului decizional mobilitate şi autonomie de aproximativ 4 ore.

Procesoarele din familia OMAP suportă sisteme de operare de nivel înalt că: Linux, Windows CE şi Android. Modulul Overo Air oferă suport pentru instalarea următoarelor sisteme de operare: Yocto Project, Ubuntu, dezvoltat de Linaro şi Angstrom. The Yocto Project

(https://www.yoctoproject.org/) permite crearea unor distribuţii Linux personalizate pentru sisteme integrate. Linaro (http://www.linaro.org/) este o organizaţie care dezvoltă build-uri de Ubuntu şi Android pentru platformele ARM. Angstrom (http://www.angstrom-distribution.org/) este o distribuţie Linux pentru dispozitive integrate.

Sistemul de operare folosit pentru dezvoltarea produsului inteligent este Ubuntu, dezvoltat de Linaro Project. Instalarea sistemului de operate pe modulul decizional presupune următorii paşi:

- Descarcarea unei imagini pre-built. Exista 3 optiuni de imagini valabile: Yocto Project, Ubuntu, Angstrom, iar imaginea de Android este in curs de dezvoltare
- Crearea unui card microSD boot-abil (pentru bootarea sistemulu Gumstix)
- Bootarea sistemului Gumstix

Pentru dezvoltarea sistemului multi-agent este necesară instalarea tool-urilor: Java Platform JDK pentru rularea aplicaţiilor Java şi JADE (http://jade.tilab.com/) pentru dezvoltarea sistemului multi-agent . Versiunea Java corespunzatoaremodulelor Overo Air care folosesc că sistem de operare Ubuntu este JDK for ARM (Linux ARM v6/v7 Hard Float ABI http://www.oracle.com/technetwork/java/javase/downloads/jdk7-arm-downloads-2187468.html), versiune pentru platformele ARM care conţine toate tool-urile software necesare dezvoltării aplicaţiilor Java pe dispozitive integrate.

Imaginea pre-built de Linux conţine suport pentru conexiunea Wifi. Configurarea conexiunii Wi-fi pentru modulul Overo presupune setarea parametrilor de reţea pentru reţeaua wireless locală. Verificarea conexiunii se face prin testarea următoarelor instrucţiuni din linie de comandă:

```
root@overo:~# iwconfig wlan0 essid any
root@overo:~# ifconfig wlan0 up
root@overo:~# iwlist wlan0 scan
```

Ultima instructiune afiseaza toate retelele Wifi vizibile. Urmatorul pas presupune modificarea fisierului pentru configurarea interfetelor de retea. Fisierul */etc/network/interfaces* se modifica prin comanda:

`vi /etc/network/interfaces` si se adauga:

```
allow-hotplug wlan0
iface wlan0 inet dhcp
      pre-up wpa_supplicant -Dwext -iwlan0 -c/etc/wpa_supplicant.conf -B
      down killall wpa_supplicant
```

Apoi se creeaza fişierul */etc/wpa_supplicant.conf* cu următorul conţinut:

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
eapol_version=1
ap_scan=1
fast_reauth=1

network={
      ssid="nume retea"
      proto=WPA2                # try WPA RSN if you WPA2 fails
      key_mgmt=WPA-PSK
      pairwise=CCMP TKIP
      group=CCMP TKIP
      scan_ssid=1
      psk="parola"
```

```
        priority=10
}
```

Conectarea la reteaua wireless se face prin inchiderea si deschiderea retelei wireless, prin urmatoarele instructiuni:

```
# ifdown wlan0
# ifup wlan0
```

Ciclul de viaţă al produsului inteligent

Ciclul de viaţă al produsului inteligent (Fig.1.3) începe cu etapă de Agregare în care este creat produsul inteligent, adică sunt agregate cele trei componente: modulul de transport, modulul decizional şi este asignat produsul ce va fi fabricat.

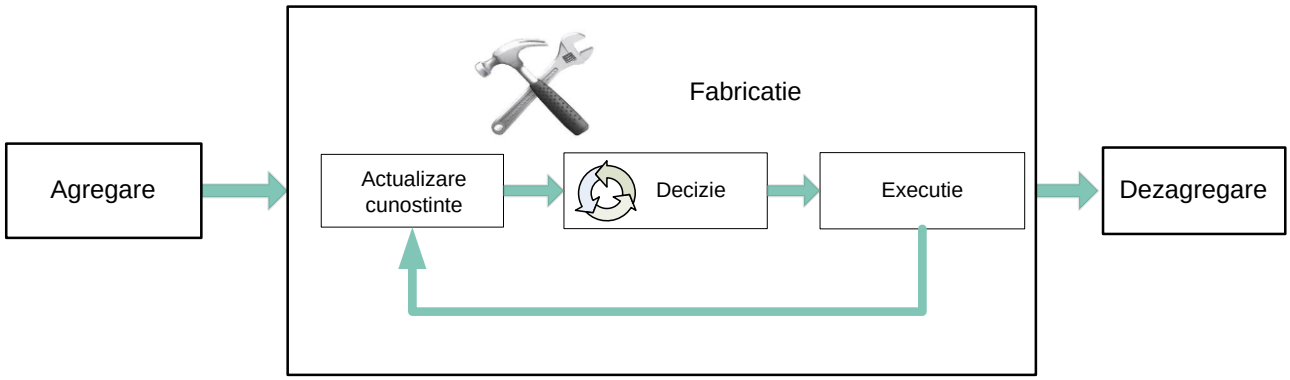


Fig. 1.3 Ciclul de viaţă al produsului inteligent

Următoarea etapă este cea de *Fabricaţie* propriu-zisă care este compusă din urmatoarele etape:

- Actualizare cunostinte – produsul inteligent actualizeaza informatiile primite de la ceilalti agenţi, Produs (secventa de operatii necesare si alte informatii legate de procesul de fabricatie) si Resursa (lista resurselor valide si incarcarea resurselor)
- Decizie – in urma dialogului cu agenţii de tip Resursa, produsul inteligent finalizeaza alocarea operatiilor pe resurse
- Executie – produsul inteligent viziteaza toate resursele selectate pentru executie pana produsul fizic este finalizat

Ultima etapă este cea de *Dezagregare*. In momentul in care produsul fizic este finalizat, produsul inteligent iese din sistem si este resetat(reciclat) pentru a executa urmatorul produs.

1.2. Structura programelor software

1.2.1. *Platforma JADE*

Pentru implementarea sistemului multi-agent este folosită platforma JADE (Java Agent DEvelopment Framework http://jade.tilab.com/). JADE este o platforma software implementată în limbajul de programare Java care oferă un middleware pentru dezvoltarea aplicaţiilor de tip multi-agent . Platforma JADE este conformă specificaţiilor FIPA (Foundation for Intelligent Physical Agents http://www.fipa.org/) – organizaţie pentru dezvoltarea standardelor software privind tehnologiile bazate pe agenţi şi interoperabilitatea acestora cu alte tehnologii.

(Bellifemine et.al.,2007) oferă documentaţia completă pentru implementarea sistemelor multi-agent folosind JADE. Facilităţile pe care platforma JADE le oferă sunt:

- un sistem distribuit in care sunt executati agenţii, fiecare agent este executat ca un thread si exista posibilitatea de a executa agenţi separat pe mai multe masini, indiferent de sistemul de operare
- platforma este conforma specificatiilor FIPA
- transport eficient al mesajelor asincrone
- implementarea paginilor aurii
- gestionarea ciclului de viata al agenţilor. In momentul in care un agent este creat, i se atribuie un identificator unic si o adresa de transport.
- Suport pentru mobilitatea agenţilor: agenţii pot migra intre masini si procese
- Instrumente grafice prin care pot fi monitorizati agenţii si platformele pe care ruleaza agenţii
- Suport pentru ontologii si limbajul de continut
- Librarie pentru protocoalele de interactiune FIPA care modeleaza diferite tipuri de comportament

Arhitectură platformei JADE este reprezentată în Fig. 5.4. şi este alcătuită din mai multe containere care pot fi distribuite în reţea. Containerele sunt procese Java care conţin toate serviciile necesare execuţiei agenţilor.

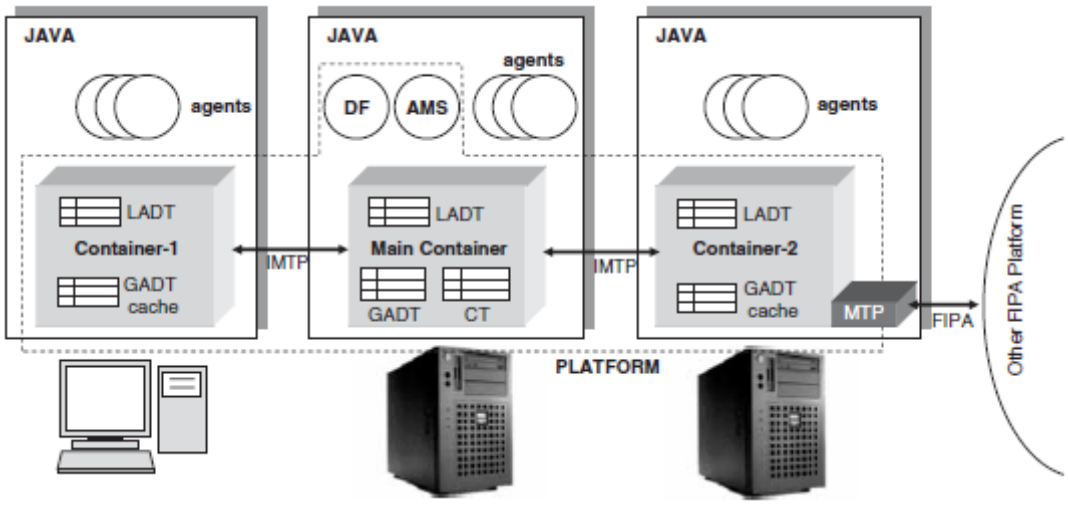


Fig.1.4. Arhitectura platformei JADE

Fiecare platforma contine un container principal numit *Main container* care este primul container care este lansat, iar celelalte containere trebuie sa se inregistreze la el.

Containerul principal are urmatoarele responsabilitati:

- Gestiunea tabelului de containere (Container Table CT):contine referintele si adresele de transport ale containerelor
- Gestiunea tabelului de descriptori a agenţilor(GADT): contine toti agenţii prezenti in platforma, statusul si locatia lor

- Lansarea agenţilor AMS(Agent Management System) si DF(Directory Facilitator) care ofera serviciile pentru gestiunea agenţilor si pagini aurii.

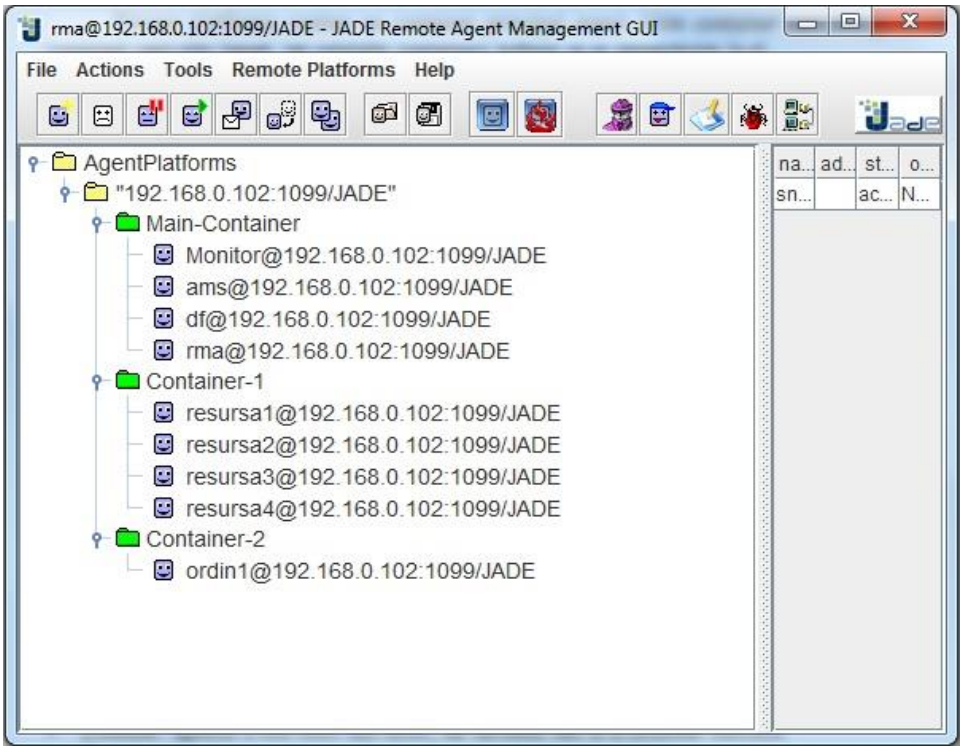


Fig. 1.5 Remote Management Agent (RMA) – consola grafică

Agentul AMS coordonează şi monitorizează întreagă platforma. AMS este folosit de către agenţi pentru a caută alţi agenţi sau pentru a-şi gestiona ciclul de viaţă. Înregistrarea agenţilor la AMS se face automat de către platforma JADE.

Agentul DF este un serviciu de tip paginii aurii şi este folosit de orice agent care doreşte să îşi înregistreze serviciile sau să caute servicii valabile. DF oferă posibilitatea subscrierilor pentru agenţii care vor să fie notificaţi în cazul în care un eveniment se petrece. Există posibiltatea lansării mai multor agenţi DF pentru a distribui serviciul de pagini aurii.

Agentul RMA – Remote Management Agent (Fig.1.5) este lansat în momentul lansării aplicaţiei şi reprezintă consola grafică prin care se gestionează activitatea agenţilor din platforma. Funcţionalităţile oferite de RMA sunt: crearea sau distrugerea agenţilor, modificarea stării agenţilor (suspendare, reactivare, distrugere), trimitere mesaje către agenţi, mutarea agenţilor în alte containere. Agentul RMA permite afişarea interfeţei grafice pentru agentul Directory Facilitator unde pot fi văzute serviciile oferite de agenţi şi lansarea Sniffer-ului în care poate fi urmărit schimbul de mesajele dintre agenţi.

Ciclul de viaţă al agenţilor din platforma JADE este conform FIPA. Un agent JADE trece prin următoarele stări:

- Initiat: agentul a fost creat, dar nu a fost inregistrat inca la AMS
- Activ: agentul a fost inregistrat, are un nume si poate comunica cu alti agenţi
- Suspendat: agentul a fost oprit pentru ca threadul care executa agentul a fost suspendat
- In asteptare: agentul asteapta un eveniment
- Eliminat: agentul a fost sters din AMS, iar threadul sau si-a terminat executia
- Transit: agentul este mutat in alta locatie

Conform specificaţiilor FIPA, platformele multi-agent trebuie să furnizeze Message Transport Service (MTS), serviciul responsabil pentru schimbul de mesaje în cadrul platformei sau între platforme, pentru a creşte interoperabilitatea sistemului. JADE implementează toate standardele Message Transport Protocol (MTP) dezvoltate de FIPA. La initializarea containerului principal, JADE porneşte un MTP bazat pe HTTP, care creează un socket server pentru containerul principal şi care apoi aşteaptă conexiuni. MTP are rolul de a ruta mesajele primite prin alte conexiuni către agenţii destinatari.

Interacţiunea dintre agenţi se face prin mesage FIPA-ACL. Structura unui mesaj FIPA este următoarea:

- Continutul mesajului
- Parametrii mesajului
- Codificarea mesajului
- Informatii de transport

Parametrii mesajului pot fi: performative, sender, receiver, content, language, ontology, protocol, conversation-id etc. Un exemplu al unui mesaj FIPA-ACL este:

```
(request
        :sender (agent-identifier :name alice@mydomain.com)
        :receiver (agent-identifier :name bob@yourdomain.com)
        :ontology travel-assistant
        :language FIPA-SL
        :protocol fipa-request
        :content
                ""((action
                (agent-identifier :name bob@yourdomain.com)
                (book-hotel :arrival 15/10/2006
                :departure 05/07/2002 ... )))"")
```

Specific mesajelor FIPA-ACL este definirea actului de comunicare. Actul de comunicare defineşte tipul de interacţiune dintre agenţi. Acestea pot fi: propose, accept proposal, agree, cancel, inform, request, query if, reject proposal, not understood etc.

Pentru definirea interacţiunii dintre agenţii din platforma JADE este folosit protocolul Contract Net Protocol.

În JADE, fiecare agent este implementat sub formă unei clase. Mai mulţi agenţi de acelaşi tip pot fi lansaţi în acelaşi timp prin lansarea mai multor instanţe ale aceleaşi clase. Taskurile pe care un agent trebuie să le execute sunt definite sub formă de comportamente (behaviours) care pot fi executate în paralel. Comportamentele sunt metode prin care se defineşte reacţia agentului când un anumit eveniment are loc, spre exemplu primirea unui anumit mesaj.

Există mai multe tipuri de comportamente:

- Comportamente simple
- Comportamente ciclice care sunt executate atat timp cat agentul este activ. Acest tip de comportament se foloseste pentru primirea mesajelor
- Comportamente singulare care sunt executate o singura data
- Comportamente paralele care contin un set de sub-comportamente care se executa in paralel
- Comportamente secventiale care contin un set de sub-comportamente care se executa unul dupa celalalt

1.2.2. *Programe agenţi*

Implementarea sistemului multi-agent presupune dezvoltarea celor 3 agenţi principali: Ordin, Produs si Resursă sub forma de clase Java.

Agentul de tip Ordin

Pentru fiecare mod de execuţie există un agent de tip ordin. Diagramele de stare sunt reprezentate în Fig. 1.6. pentru modul secvenţă completă, respectiv 1.7. pentru modul următoarea operaţie. În cele două diagrame se poate observă că comportamentul agentului Ordin este divizat în 3 etape: Actualizare cunoştinţe, Decizie şi Execuţie, etape care se regăsesc în ciclul de viaţă al produsului inteligent (Fig. 1.3.).

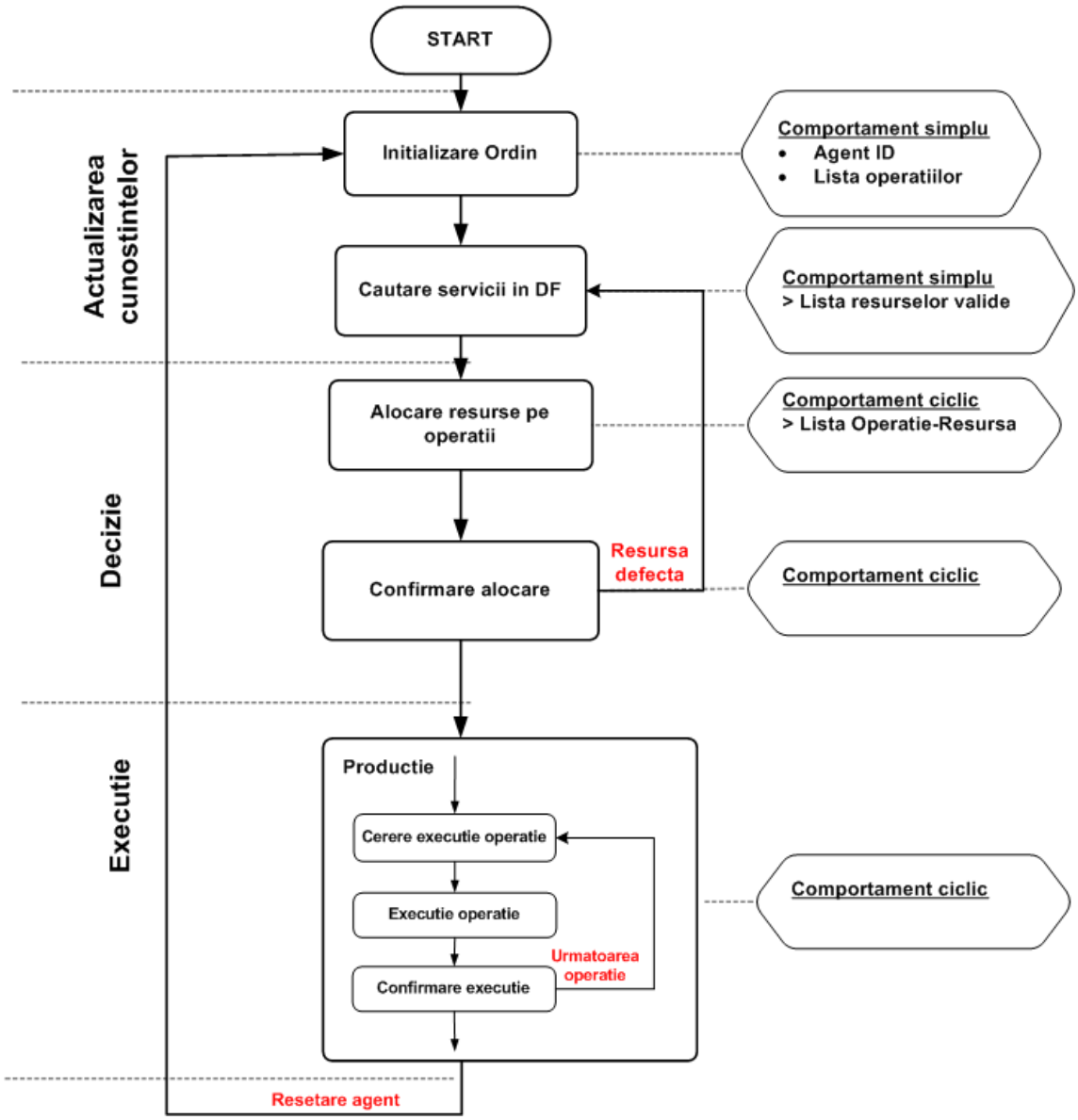


Fig. 1.6. Diagrama de stare a agentului Ordin – Cazul 1: secventa completa

Procesul de fabricaţie al produselor este implementat printr-un comportament secvenţial format dintr-un set de sub-comportamente care sunt executate secvenţial.

```
SequentialBehaviour seqBehaviour = new SequentialBehaviour();
Behaviour receiveOperationList = new receiveOperationListBehaviour(this);
Behaviour searchServicesDF = new searchServicesDFBehaviour(this);
Behaviour updateKnowledge = new UpdateKnowledgeBehaviour(this);
Behaviour sendProposeMessage = new sendProposeMessageBehaviour(this);
Behaviour receiveAcceptMessage = new receiveAcceptMessageBehaviour(this);
Behaviour palletCellEntry = new PalletCellEntryBehaviour(this);
Behaviour productionExecution = new productionExecutionBehaviour(this);
```

In etapa de *Actualizare a cunostintelor*, agentul ordin este format din urmatoarele comportamente:

- *receiveOperationListBehaviour* – in care agentul primeste informatii (lista de operatii) de la agentul Produs;
- *searchServicesDFBehaviour* – agentul cauta in Directory Facilitator resursele valide pentru lista de operatii; in urma cautarii rezultatul este lista de resurse valide pentru fiecare operatie. Apoi, agentul Ordin trimite mesaje de tip *QUERY_REF* catre resursele din lista rezultata.

Sintaxa de cautare a serviciilor in Directory Facilitator este urmatoarea:

```
DFAgentDescription dfd = new DFAgentDescription();
ServiceDescriptionsd  = new ServiceDescription();
sd.setType(numeOperatie);
dfd.addServices(sd);
DFAgentDescription[] result = DFService.search(this, dfd);
```

Etapa de *Decizie* este formata din comportamentele:

- *updateKnowledgeBehaviour* – este un comportament ciclic in care agentul primeste de la resurse mesaje de tip *INFORM* ce au in continut incarcarea resursei si incepe procesul de alocare a resurselor pe operatii. Agentul de tip Ordin alege pentru fiecare operatie resursa valida cu incarcarea cea mai mica.

Algoritmul folosit in procesul de alocare este urmatorul:

```
startBehaviourTime = System.currentTimeMillis();
ACLMessage receivedMessage =
this.myAgent.receive(MessageTemplate.MatchPerformative(ACLMessage.INFORM));

if(receivedMessage!= null)
{
orderAgent.setOrderState("Scheduling");
orderAgent.sendMessage(new
AID("Monitor",AID.ISLOCALNAME),orderAgent.getOrderState(),ACLMessage.INFORM,null);
responseNo++;
for(int i=0;i<orderAgent.getOrderOperations().length;i++)
{
```

```
if(orderAgent.getOrderOperations()[i].equals(receivedMessage.getConversationId()))
{
if(Integer.parseInt(receivedMessage.getContent())<resourceLoad[i])
{
resourceLoad[i] = Integer.parseInt(receivedMessage.getContent());
chosenResource[i] = receivedMessage.getSender().getLocalName();
chosenOperation[i] = receivedMessage.getConversationId();
}
}
}
if(responseNo == orderAgent.getSearchResourceNo() || (System.currentTimeMillis()-
startBehaviourTime > 5000))
{
orderAgent.setChosenOperation(chosenOperation);
orderAgent.setChosenResource(chosenResource); }
```

Comportamentul ciclic este finalizat dacă au fost primite toate mesajele sau dacă au trecut 5 secunde de la începutul execuţiei comportamentului. Dacă numărul mesajelor de tip INFORM primite este mai mic decât numărul mesajelor de tip QUERY_REF înseamnă că unele resurse nu sunt funcţionale, posibil datorită unei defectări. Aceste resurse nu vor fi luate în considerare în procesul de alocare.

- *sendProposeMessageBehaviour* – comportament de tip simplu in care ordinul trimite mesaje de tip *PROPOSE* resurselor alocate
- *receiveAcceptMessageBehaviour* – comportament de tip ciclic in care agentul de tip ordin primeste mesaje de tip *ACCEPT_PROPOSAL* de la resursele care accepta alocarea. Dupa confirmarea executiei, resursele si operatiile efectuate sunt inserate intr-o structura de tip ArrayList folosita mai apoi la executia operatiilor.

`ArrayList<ResourceOperationList> arrayList = new ArrayList<ResourceOperationList>(),` unde *ResourceOperationList* are urmatoarele atribute:

```
String operation;
String resource;
```

Comportamentul ciclic este finalizat dacă au fost primite toate mesajele sau dacă au trecut 5 secunde de la începutul execuţiei comportamentului. În cazul în care nu au fost trimise toate mesajele, se reia procesul de căutare a serviciilor în DF pentru operaţiile care au rămas nealocate şi, mai apoi, se execută alocarea acelor operaţii. Re-execuţia acestor comportamente se face prin resetarea comportamentului secvenţial, schimbând parametrii de căutare a operaţiilor.

Următoarea etapă, de Execuţie, conţine comportamentele:

- *productionExecutionBehaviour* care este un comportament de tip simplu in care agentul seteaza variabilele OPC din automatul programabil pentru a executa transportul paletelor la posturile de lucru.

Comunicaţia cu modulul TCP-OPC bridge se face prin intermediul protocolului TCP/IP. Agentul de tip ordin conţine clientul TCP/IP care comunica cu serverul TCP/IP aflat în modulul TCP-OPC bridge. Pentru conectarea clientului la socketul deschis de serverul TCP/IP şi verificarea conexiunii se foloseşte următoarea sintaxă:

```
Socket clientSocket = new Socket(adresaIP, adresaPort)
if(clientSocket.isBound()){
System.out.println("Bind to socket: "+clientSocket.getInetAddress()+ "
port:"+clientSocket.getPort());}
```

Citirea şi scrierea se fac prin:

```
try
{
inFromServer = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
inputServer = inFromServer.readLine();
}
catch(IOException ioe)
{
ioe.printStackTrace();
}

outToServer = new PrintWriter(clientSocket.getOutputStream());
outToServer.print(a);
outToServer.flush();
```

Cele două aplicaţii comunica prin intermediul a trei tipuri de mesaje:

1. Cerere locaţie port-paletă în celulă de fabricaţie: agentul de tip Ordin interoghează variabilele PLC pentru a afla poziţia produsului inteligent în sistemul de transport

Formatul mesajului este:00 0i 01 06, unde i este identificatorul ordinului şi al port-păleţei, secvenţă 01 reprezintă tipul mesajului şi ultima secvenţă 06 este lungimea mesajului.

Răspunsul dat de modulul TCP-OPC bridge este:

00 0i 01 ll xxx ... xxx yyy...yyy, unde ll–lungimea mesajului,xxx – reprezintă valoarea variabilelor location_pal_1-5 din automatul programabil şi semnifică identificatorul segmentului din bandă transportoare unde se află paletele 1-5 şi yyy – reprezintă valoarea variabilelor paleta_în_r1-4, adică identificatorul paletelor care se află la cele 4 posturi de lucru. Pentru a verifică dacă o paleta a ajuns la un post de lucru, se compară câmpurile xxx şi yyy, astfel încât identificatorul păleţei dintr-un anumit post de lucru să coincidă cu ID-ul păleţei care se află la acel post de lucru.

2. Cerere scriere variabile resurse ce urmează a procesa produsul: după alocarea operaţiilor pe resurse, agentul Ordin transmite automatului programabil lista resurselor către care dispozitivele port-paleta trebuie transportate.

Formatul mesajul este: 00 0i 0254 xxx xxx xxx xxx .....xxx, unde i este identificatorul port-păleţei, iar xxx este numărul postului de lucru la care va fi transportată port-paleta. Numărul maxim de operaţii care pot fi executate este 16. La primirea mesajului, modulul TCP-OPC scrie variabilă de tip vector de structuri şir_paleta. Vectorul are dimensiunea 16, iar structura are următoarele componente: post, operaţie, timpmin, timpmax şi raport.

Pentru a evacua paleta din sistemul de transport la sfârşitul producţiei, se scrie după ultimul post de lucru ce trebuie vizitat, valoarea 255 în dreptul postului.

3. Cerere schimbare stare variabilă robot, agentul Ordin informează automatul programabil în legătură cu finalizarea operaţiei pentru că paleta să fie transportată către postul de lucru următor.

Formatul mesajului este: 00 0i 0250 x y, unde i este identificatorul port-păleţei, x este identificatorul resursei şi y este starea – true sau false.

După actualizarea variabilelor OPC, automatul programabil începe transportul paletelor la posturile de lucru. Agentul ordin interoghează la o perioada de 500 milisecunde poziţia păleţei în sistem prin trimiterea mesajelor de tip 1 – cerere locaţie port-paleta. În momentul în care paleta a ajuns în dreptul resursei, agentul ordin trimite mesaj de tip REQUEST, , însoţit de identificatorul operaţiei, către resursă pentru a cerere începerea execuţiei şi se trece la următorul comportament.

- *receiveConfirmMessageBehaviour,* comportament de tip ciclic în care se aşteaptă primirea mesajelor de tip CONFIRM de la resursă. Primirea unui astfel de mesaj reprezintă finalizarea operaţiei, iar ordinul trimite către modulul TCP-OPC bridge un mesaj de tip 3 – cerere schimbare stare robot prin care semnalează că operaţia a fost efectuată, iar paleta poate fi transportată către următorul post de lucru. După finalizarea comportamentului se trece la comportamentul anterior, productionExecutionBehaviour şi se execută următoarea operaţie.

După finalizarea execuţiei ultimei operaţii, port-paleta este evacuată din sistem şi poate fi refolosită pentru fabricarea altui produs.

Diferenţa dintre cele două moduri de execuţie este că în cazul secvenţei complete, alocarea resurselor se face de la început pentru toată lista de operaţii, în timp ce în cazul următoarei operaţii, alocarea resurselor se face doar pentru următoarea operaţie. Astfel, după finalizarea execuţiei operaţiei curente, se reia procesul de căutare a resurselor valide în Directory Facilitator şi de alocare a resurselor.

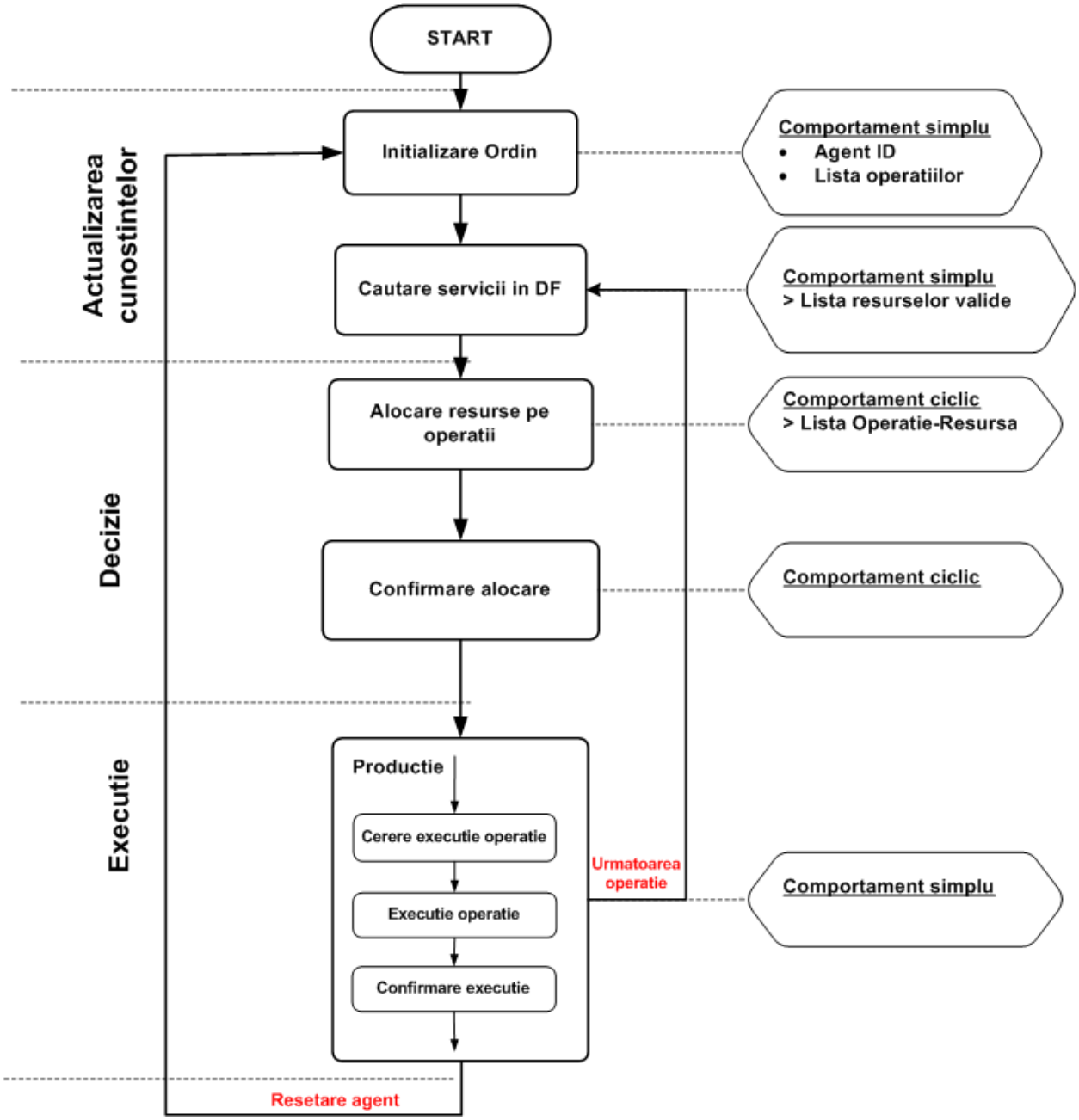


Fig. 1.7. Diagrama de stare a agentului Ordin – Cazul 2: următoarea operaţie

Agentul de tip Resursă

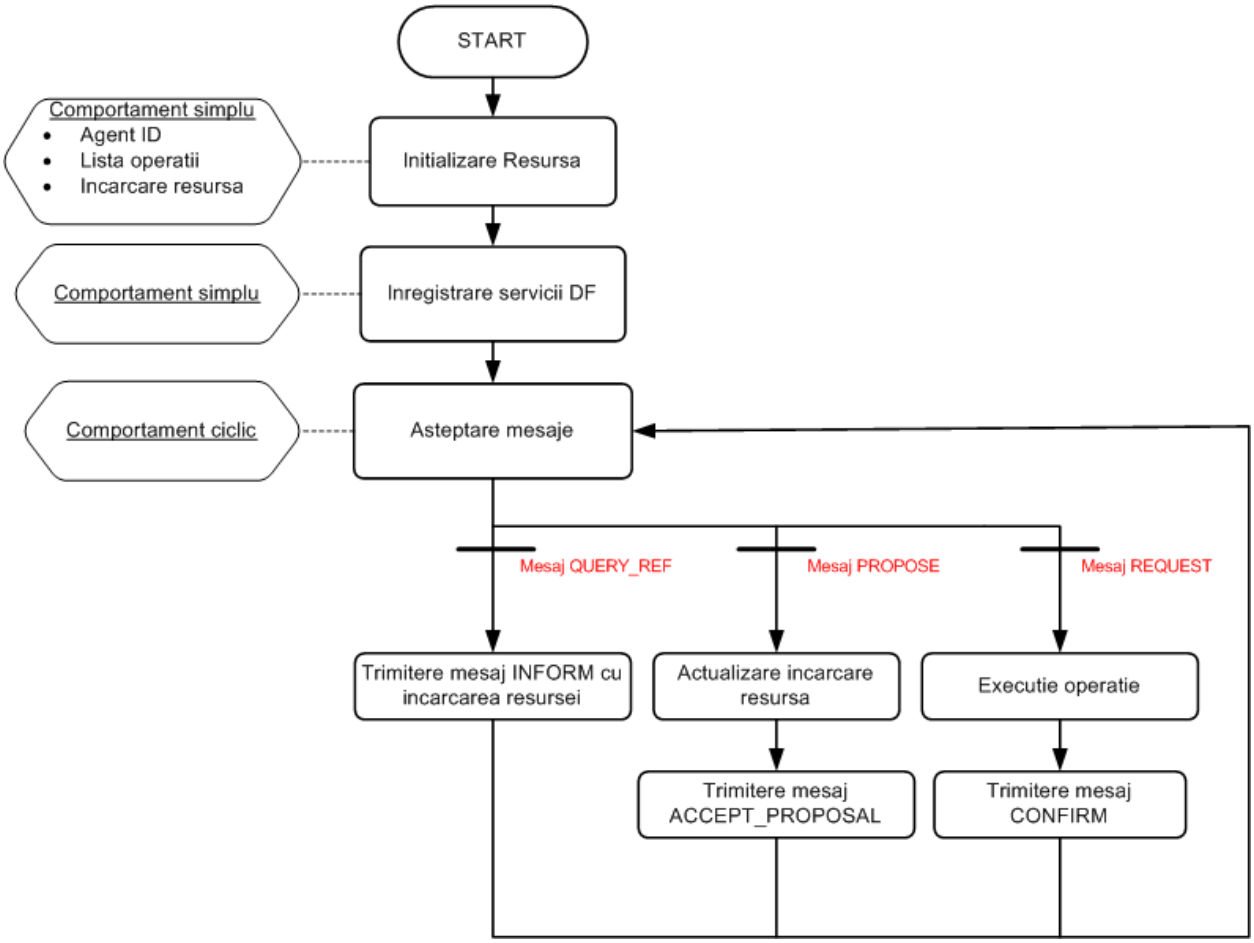


Fig. 1.8. Diagrama de stare a agenţilor de tip Resursă

Comportamentul agentului de tip Resursă începe cu etapă de initializare în care sunt setate ID-ul agentului, capabilităţile resursei (lista de operaţii posibile) şi încărcarea resursei. După etapă de initializare urmează etapă de înregistrare a serviciilor la Directory Facilitator (serviciul de pagini aurii). În această etapă fiecare resursă înregistrează toate operaţiile posibile. Înregistrarea unui serviciu la DF se face în modul următor:

```
DFAgentDescription dfd = new DFAgentDescription();
dfd.setName(getAID() );
ServiceDescription sd  = new ServiceDescription();
sd.setType(numeOperatie);
sd.setName(numeOperatie);
dfd.addServices(sd);

try {DFService.register(this, dfd );}
catch (FIPAException fe) { fe.printStackTrace(); }
```

Urmează un comportament ciclic în care Resursa aşteaptă mesaje de la agentul de tip ordin. Pentru citirea mesajelor primate se foloseşte sintaxa:

```
ACLMessage receivedMessage = receive();
```

Există 3 tipuri de mesaje pe care agentul Resursa le primeste, în funcţie de actul de comunicare:

- Mesaje de tip *QUERY_REF* – in acest caz agentul Ordin trimite o cerere pentru incarcarea resursei catre resursa, iar resursa raspunde cu un mesaj de tipul *INFORM* care are in continut incarcarea resursei

- Mesaje de tip *PROPOSE* – acest tip de mesaj semnifica faptul ca resursa a fost aleasa in procesul de alocare a resurselor pentru a efectua o anumita operatie. Dupa primirea acestui mesaj, incarcarea resursei va fi incrementata cu 1 si resursa va trimite inapoi mesaj de tip *ACCEPT_PROPOSAL*
- Mesaje de tip *REQUEST* – in acest caz agentul Ordin trimite cerere pentru executia operatiei catre resursa. Resursa incepe executia operatiei si dupa finalizarea acesteia trimite un mesaj de *CONFIRM.*

Selectia mesajelor dupa tipul actului de comunicare se face utilizand urmatoarea sintaxa:

```
if(receivedMessage!= null)
{
        switch(receivedMessage.getPerformative())
        {
                case(ACLMessage.QUERY_REF):
                {
                // code
                }break;
        }
}
```

Schimbul de mesaje ce are loc intre agenţii de tip Ordin si cei de tip Resursa poate fi vizualizat in Fig. 1.9.

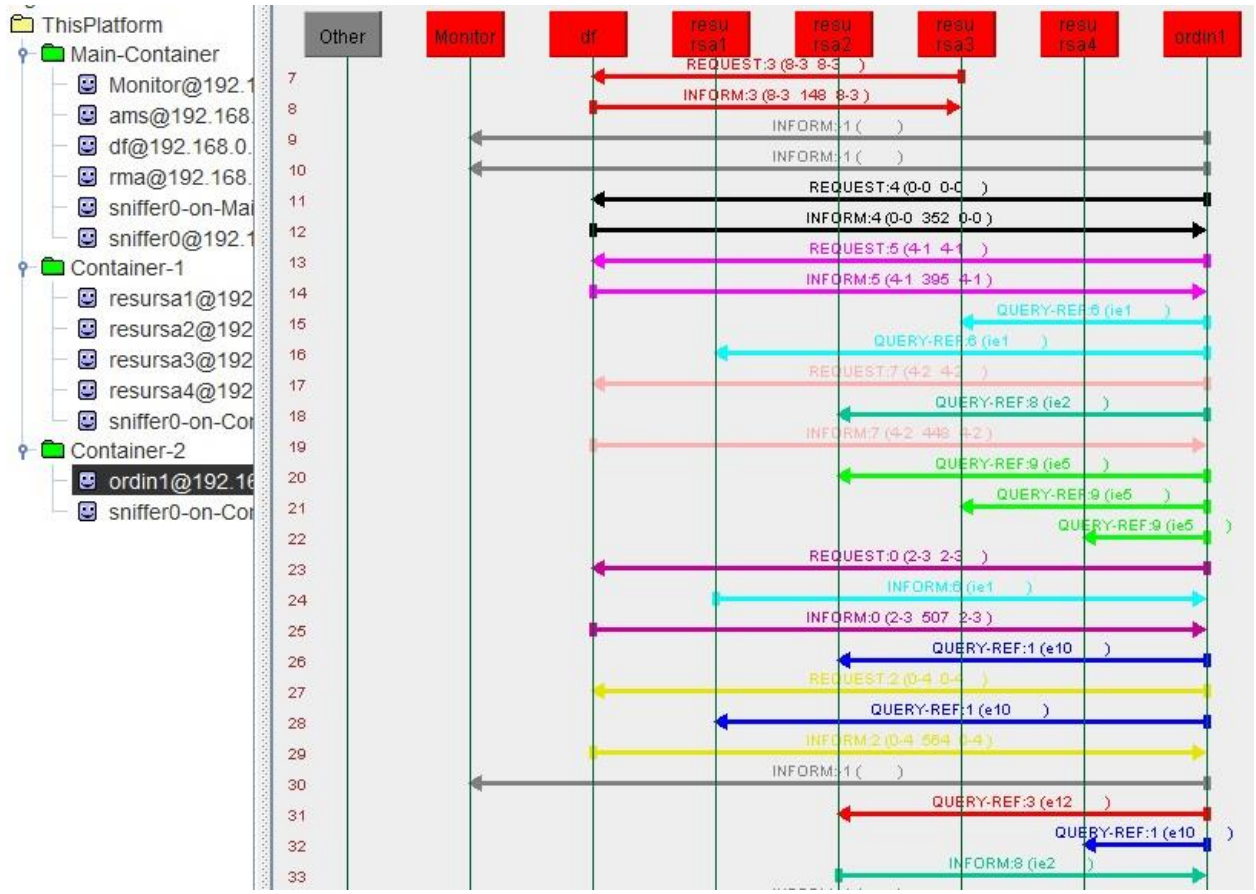


Fig.1.9. Schimbul de mesaje între agenţii Ordin şi agenţii Resursă (Sniffer)

1.2.3. *Modulul TCP-OPC bridge*

Comunicaţia dintre agenţii de tip Ordin şi automatul programabil se face prin intermediul unui protocol numit TCP-OPC bridge.

Modulul TCP-OPC este o aplicaţie software dezvoltată în limbajul de programare C#. Această aplicaţie conţine un Server TCP/IP care comunica prin socket cu Clientul TCP/IP dezvoltat în aplicaţia multi-agent şi un Client OPC care comunica cu Serverul OPC instalat în automatul programabil. Clientul OPC este responsabil pentru scrierea şi citirea datelor de intrare şi ieşire din automatul programabil prin intermediul elementelor OPC (item-uri şi grupuri).

Interacţiunea dintre aplicaţia de tip agent şi TCP-OPC bridge se face printr-un schimb de mesaje utilizând Comunicaţia prin socket şi protocolul de transport TCP/IP.

Comunicaţia dintre TCP-OPC bridge este realizată prin intermediul OPC. OPC (Object Linking and Embedding for Process Control) este un standard de comunicaţie în timp-real care permite interacţiunea dintre dispozitivele industriale hardware care provin de la producători diferiţi.

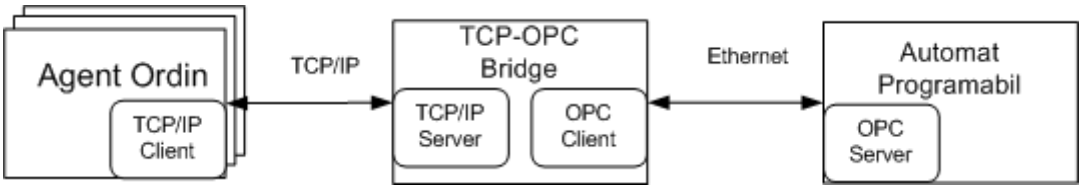


Fig. 1.10. Conexiunea dintre agenţii Ordin şi automatul programabil prin TCP-OPC bridge

Scrierea si citirea variabilelor din automatul programabil se face utilizand libraria *OPCdotNETLib.dll*. Pasii pentru accesarea variabilelor din automat sunt:

- Definirea serverului OPC: `string serverProgID = "IndraLogic.OPC.02"`
- Definirea variabilelor automat : `string itemA = ".location_pal_1"`
- Conectarea la serverul OPC
  ```
  OpcServer opcServer = new OpcServer();
  opcServer.Connect(serverProgID);
  ```
- Definirea si inregistrarea grupurilor
  ```
  OpcGroup opcGroup = opcServer.AddGroup("Grup1", false, 200);
  ```
- Definirea si inregistrarea item-urilor:
  ```
  OpcItemDef itemDefs = new OPCItemDef(itemA, true, 0, VarEnum.VT_EMPTY);
  ```
- Adaugarea item-urilor la grupuri:
  ```
  opcGroup.AddItems(itemDefs, out rItm);
  ```
- Citirea si scrierea item-urilor se executa folosind functiile `OPCWriteGroup.Write()` si `OPCWriteGroup.Read()`.

Serverul Tcp/Ip deschide un socket pentru fiecare agent de tip Ordin. Crearea unui socket şi acceptarea clienţilor Tcp/Ip se fac prin următoarea sintaxă:

```
try{
TcpListener listener = new TcpListener(IPAddress.Any, TcpPort);
listener.Start();
Socket s = listener.AcceptSocket();
}
catch  {System.Windows.Forms.MessageBox.Show("Eroare creare socket acceptare conexiune");}
```

2. Scenarii experimentale

2.1. Scenarii propuse

Pentru testarea şi validarea soluţei de conducere sunt folosite următoarele scenarii:

- Simulare proces de fabricaţie utilizând sistemul mulţi-agenţi şi
- Execuţia fabricaţiei pe sistemul fizic dintr-o celulă pilot cu resurse industriale multiple (roboti industriali, masini CNC, camere video), in care se pot executa procese de productie de tip job-shop.

Primul scenariu implică execuţia aplicaţiei într-un mediu fără perturbatii. Cel de-al doilea scenariu presupune apariţia unei perturbatii de tip defectarea unei resurse în timpul procesului de fabricaţie.

Fiecare scenariu va fi testat pentru cele două moduri de execuţie: secvenţă completă şi următoarea operaţie. Pentru testarea şi validarea soluţiei se vor compară timpii de execuţie obţinuţi pentru fiecare scenariu.

2.2. Modul de lansare în execuţie a aplicaţiei

Lansarea în execuţie a aplicaţiei presupune următorii paşi:

- Pornirea automatului si a aplicatiei OPC Server
- Lansarea modulului TCP-OPC bridge
- Atribuirea identificatorului unic produselor inteligente, adica scrierea RFID-urilor pe port-palete
- Executia agenţilor de tip Resursa pe statiile PC asociate. Pentru lansare se folosesc fisierele script asociate sau se lanseaza proiectul din IDE-ul Eclipse

Fişierul script contine urmatoarea sintaxa:

```
java jade.Boot -container
Resursa1:ResourceAgent.ResourceAgent
pause
```

- Executia agenţilor de tip Ordin pe modulul decizional Overo. Pentru lansare se executa fisierele *OrderAgent.sh*:

```
#!/bin/bash
echo "Hello World"
java jade.Boot -container -host 192.168.0.10 -port 1099 -local-host
192.168.0.61 -local-port 1099 ordin1:OrderAgent.OrderAgent
```

- Se plaseaza port-paletele pe sistemul de transport si se asteapta finalizarea productiei
- Dupa finalizarea executiei produselor, port-paletele sunt reinitializate pentru a fi reutilizate in productia urmatoarelor produse.

2.3. Analiza rezultatelor obţinute şi validarea soluţiei propuse

Testarea soluţiei de conducere presupune compararea celor două moduri de execuţie: secvenţă completă şi următoarea operaţie.

În primele două cazuri (realizarea unui singur ordin – Tabelul 2.1. şi realizarea unui lot de 4 produse – Tabelul 2.2.) au fost simulate procesele de fabricaţie. Timpul de transport al port-paletelor în celula de fabricaţie şi execuţia operaţiilor pe resurse au fost simulate pentru 5 secunde.

Mod de executie	Fara perturbatii	Cu perturbatii
Secventa completa	00:41	01:02
Urmatoarea operatie	00:41	00:41

Tabel 2.1. Cazul 1: simulare fabricaţie pentru ordinul 1

Se poate observa în cele două cazuri că primul mod de execuţie oferă un timp de fabricatie mai bun în cazul în care în sistem nu apar perturbatii, însă timpul de execuţie este mai mare decât cel de-al doilea mod de execuţie în cazul apariţiei perturbatiilor.

Mod de executie	Fara perturbatii	Cu perturbatii
Secventa completa	02:01	02:55
Urmatoarea operatie	02:12	02:24

Tabel 2.2. Cazul 2: simulare fabricaţie pentru întreaga producţie – 4 ordine

Pentru următoarele două cazuri, timpul total de execuţie a fost măsurat pentru execuţia pe sistemul fizic. Se poate observa că în cazul fabricaţiei unui singur produs – tabelul 2.3., timpul de execuţie pentru modul *Următoarea operaţie* este mai mare decât cel pentru *Secvenţa completă* în situaţia fără perturbaţii, însă în cazul apariţiei perturbaţiei, timpul de execuţie pentru modul *Secvenţa completă* creşte, în timp ce cel pentru modul *Următoarea operaţie* rămâne acelaşi.

Mod de executie	Fara perturbatii	Cu perturbatii
Secventa completa	01:41	01:55
Urmatoarea operatie	01:46	01:46

Tabel 2.3. Cazul 3: execuţie fabricaţie pentru ordinul 1

În cazul realizarii unui lot de 4 produse, tabelul 2.4, modul de execuţie *Următoarea operaţie* oferă performanţe mai bune atât în cazul fără perturbatii, cât şi în cazul cu aparitiei de perturbatii.

Acest lucru poate fi datorat alocării operaţiilor pe resurse care a făcut ca transportul port-paletelor între posturile de lucru să fie mai scurt.

Mod de executie	Fara perturbatii	Cu perturbatii
Secventa completa	06:17	06:39
Urmatoarea operatie	05:06	05:30

Tabel 2.4. Cazul 4 : execuţie fabricaţie pentru întreaga producţie – 4 ordine

Timpul total de executie mai mare in cazul cu perturbatii decat in cel fara perturbatii este datorat timpului de realocare a operatiilor ce au ramas nealocate din cauza defectarii resursei (timpul de semnalare a perturbatiei este de 5 secunde – agentul Ordin asteapta 5 secunde pentru a primi raspuns de la agentii de tip Resursa; daca nu raspund toate resursele, este semnalata o defectare, dupa care este refacuta alocarea – cautarea in DF si alocarea resurselor pe operatii).

Timpul total de executie este egal pentru modul de executie *Urmatoarea operatie* indiferent de perturbatii, deoarece aparitia perturbatiilor nu aduce modificari in procesul de alocare (o resursa defecta nu mai apare in DF pentru alocarea urmatoarei operatii).

În urma analizei rezultatelor obţinute prin testarea soluţiei de conducere heterarhice, s-a observat că modul *Secvenţa completă* oferă performanţe mai bune în situaţia în care nu apar perturbatii, adică timpul total de execuţie al produselor în lipsa perturbatiilor este mai bun, în timp ce cel de-al doilea mod de execuţie este mai performant în cazul apariţiei perturbatiilor de tip defectare resurse.

3. Dezvoltări experimentale

3.1. Realizări

În cadrul acestei dezvoltari a fost prezentată o soluţie de conducere de tip heterarhic centrată pe fluxul de produse din sistemul de fabricaţie.

Prin conceptul de automatizare dirijată de produs se specifică faptul că produsele sunt entităţi active în procesul de fabricaţie, fiind implicate în luarea deciziilor referitoare la producţie. În cadrul proiectului, acest concept este asigurat prin utilizarea produselor inteligente. Produsul inteligent folosit în soluţia de conducere propusă are capacitate decizională şi de stocare a informaţiilor, iar inteligenţă este îmbarcată în produs.

Au fost prezentate două moduri de execuţie: în primul caz, planificarea operaţiilor şi alocarea resurselor se fac înainte de începerea execuţiei, iar în cel de-al doilea caz alocarea resurselor se face înaintea următoarei operaţii. Arhitectură sistemului de conducere şi cele două moduri de execuţie asigura robusteţe, flexibilitate şi toleranţă la defect.

Sistemul de conducere este implementat printr-un sistem multi-agent . Principalii agenţi sunt: resursă, produs şi ordin. Dezvoltarea sistemului multi-agent a fost făcută utilizând platforma JADE, iar sistemul de conducere a fost testat şi validat în celulă pilot cu resurse industriale multiple (roboti industriali, masini CNC, camere video).

3.2. Propuneri aplicare si dezvoltare viitoare

Sistemul de conducere heterarhic poate fi utilizat in mod curent pentru conucerea automata a proceselor de fabric atie de tip job-shop. Sistemul poate fi îmbunătăţit prin creşterea

complexităţii procesului de alocare a operaţiilor pe resurse. Criteriile care ar putea fi folosite in algoritm sunt:

- Costul transportului – fiecare sectiune din sistemul de transport va avea asociat un cost, iar selectia resurselor se va face astfel incat costul transportului produsului de la postul de lucru curent la postul de lucru urmator sa fie minim
- Timpul de execuţie al operaţiilor pentru fiecare resursă, în cazul în care acesta diferă
- Cel mai apropiat moment de execuţie (fereastra temporală)

O altă posibilă îmbunătăţire este calculul secvenţei optime a operaţiilor. Agentul de tip ordin este responsabil pentru planificarea operaţiilor, ţinând cont de restricţiile privind ordinea operaţiilor. După calculul tuturor secvenţelor posibile este aleasă secvenţa care oferă cel mai bun timp total de procesare.

Anexe

Agentul de tip Ordin – cazul secventiere completa

```
package OrderAgent;

import java.util.logging.*;
import java.io.*;
import java.util.*;

import TimeStamp.TimeStampFormatterUtil;
import jade.core.*;
import jade.core.behaviours.*;
import jade.util.leap.Iterator;
import jade.domain.FIPAException;
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.*;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

public class OrderAgent extends Agent {
private String orderName;
private String orderState="Initialization";
private int operationNo = 0;
private String[] orderOperations;
private String[] currentOrderOperations;
private String[] extraOrderOperations;
private int currentOperationNo = 0;
private int searchResourceNo = 0;
private Logger logger = Logger.getLogger("MyLog");
private String[] chosenOperation;
private String[] chosenResource;
private boolean resourceBreakdown = false;
private long beginExecutionTime;
private long executionTime;
protected SequentialBehaviour seqBehaviour;
```

```
protected ArrayList<ResourceOperationList> arrayList = new ArrayList<ResourceOperationList>();
private TCPClient clientSocket = new TCPClient();
public String getOrderName() {return orderName;}
public int getOperationNo() {return operationNo;}
public void setOperationNo(int operationNo) {this.operationNo = operationNo;}
public Logger getLogger() {return logger;}
public String[] getOrderOperations() {return orderOperations;}
public void setOrderOperations(String[] orderOperations) {this.orderOperations = orderOperations;}
public String getOrderState() {return orderState;}
public void setOrderState(String orderState) {this.orderState = orderState;}
public int getSearchResourceNo() {return searchResourceNo;}
public void setSearchResourceNo(int searchResourceNo) {this.searchResourceNo = searchResourceNo;}
public String[] getChosenOperation() {return chosenOperation;}
public void setChosenOperation(String[] chosenOperation) {this.chosenOperation = chosenOperation;}
public String[] getChosenResource() {return chosenResource;}
public void setChosenResource(String[] chosenResource) {this.chosenResource = chosenResource;}
public TCPClient getClientSocket() {return clientSocket;}
public ArrayList<ResourceOperationList> getArrayList() { return arrayList;}
public void setArrayList(ArrayList<ResourceOperationList> arrayList) {
                this.arrayList = arrayList;
        }
public long getBeginExecutionTime() {
                return beginExecutionTime;
        }
public void setBeginExecutionTime(long beginExecutionTime) {
                this.beginExecutionTime = beginExecutionTime;
        }
        public void setExecutionTime(long executionTime) {
                this.executionTime = executionTime;
        }
        public boolean isResourceBreakdown() {
                return resourceBreakdown;
        }
        public void setResourceBreakdown(boolean resourceBreakdown) {
                this.resourceBreakdown = resourceBreakdown;
        }
        public String[] getExtraOrderOperations() {
                return extraOrderOperations;
        }
        public void setExtraOrderOperations(String[] extraOrderOperations) {
                this.extraOrderOperations = extraOrderOperations;
        }
        public String[] getCurrentOrderOperations() {
                return currentOrderOperations;
        }
        public void setCurrentOrderOperations(String[] currentOrderOperation) {
                this.currentOrderOperations = currentOrderOperation;
        }
        public int getCurrentOperationNo() {
                return currentOperationNo;
        }
```

```
	public void setCurrentOperationNo(int currentOperationNo) {
		this.currentOperationNo = currentOperationNo;
	}

	protected void setup()
	{
		orderName = this.getLocalName();
		System.out.println("Agent "+ orderName+" launched...");
		logger.info("Order "+orderName+" launched!");

		sendMessage(new
AID("Monitor",AID.ISLOCALNAME),orderState,ACLMessage.INFORM,null);

	  clientSocket.startClientSocket(this.getLocalName());

	  try
	  {
		String filePath = "E:/Roxana/Logger/"+this.getLocalName()+"Logger.log";
	FileHandler fileHandler = new FileHandler(filePath, false); // append = true
	    logger.setUseParentHandlers(false); // no console output
	    logger.addHandler(fileHandler);
	    fileHandler.setFormatter(new SimpleFormatter());
	  } catch (Exception e)
	  {
	    e.printStackTrace();
	  }
	  seqBehaviour = new SequentialBehaviour();
	  Behaviour receiveOperationList = new receiveOperationListBehaviour(this);
	  Behaviour searchServicesDF = new searchServicesDFBehaviour(this);
		Behaviour updateKnowledge = new UpdateKnowledgeBehaviour(this);
	  Behaviour sendProposeMessage = new sendProposeMessageBehaviour(this);
	  Behaviour receiveAcceptMessage = new receiveAcceptMessageBehaviour(this);
	  Behaviour palletCellEntry = new PalletCellEntryBehaviour(this);
	  Behaviour productionExecution = new productionExecutionBehaviour(this);
	  seqBehaviour.addSubBehaviour(receiveOperationList);
	  seqBehaviour.addSubBehaviour(searchServicesDF);
	  seqBehaviour.addSubBehaviour(updateKnowledge);
	  seqBehaviour.addSubBehaviour(sendProposeMessage);
	  seqBehaviour.addSubBehaviour(receiveAcceptMessage);
	  seqBehaviour.addSubBehaviour(palletCellEntry);
	  seqBehaviour.addSubBehaviour(productionExecution);
	  addBehaviour(seqBehaviour);
	}

	public void sendMessage(AID a,String content, int message_type, String convID)
	{
		ACLMessage mesajTrimis;
		mesajTrimis = new ACLMessage(message_type);
		mesajTrimis.addReceiver(a);
		mesajTrimis.setContent(content);
		mesajTrimis.setConversationId(convID);
		this.send(mesajTrimis);
```

```
                logger.info("Sending "+ACLMessage.getPerformative(message_type)+" message to
"+a.getLocalName()+".");
        }
}
class receiveOperationListBehaviour extends OneShotBehaviour
{
        private OrderAgent orderAgent;
        private RandomAccessFile randomAccesFile;
        receiveOperationListBehaviour(OrderAgent oa)
        {
                super(oa);
                orderAgent = oa;
        }
        public void action()
        {
                if(!orderAgent.isResourceBreakdown())
                {
                        /*
                         *  Comportament in caz normal
                         */
                        String[] operations = new String[2];
                        String operationsList="List of operations:\n";

                        int i=0;
                        try
                        {
                                randomAccesFile = new
RandomAccessFile("E:/Roxana/Lista"+orderAgent.getOrderName()+".txt", "r");
                        }
                        catch(FileNotFoundException fnf)
                        {
                                fnf.printStackTrace();
                                System.exit(0);
                        }
                        FileInputStream fileStream = null;
                        String readLineString;
                        try
                        {
                                int operationNumber = Integer.parseInt(randomAccesFile.readLine());
                                orderAgent.setOperationNo(operationNumber);
                                orderAgent.setCurrentOperationNo(operationNumber);
                                System.out.println("Number of operations:" + operationNumber);

                                operations = new String[operationNumber];
                                while ((readLineString = randomAccesFile.readLine()) != null)
                                {
                                        System.out.println("" +readLineString);
                                        operations[i] = readLineString;
                                        i++;
                                        operationsList += '\t'+readLineString+'\n';
                                }
                        }
```

```
                        catch (IOException e)
                        {
                                e.printStackTrace();
                        } finally
                        {
                                try
                                {
                                        if (fileStream != null)
                                                fileStream.close();
                                }
                                catch (IOException ex)
                                {
                                        ex.printStackTrace();
                                }
                        }
                        orderAgent.setOrderOperations(operations);
                        orderAgent.setCurrentOrderOperations(operations);
                        orderAgent.getLogger().info("The order has "+orderAgent.getOperationNo()+"
operations.");
                        orderAgent.getLogger().info(operationsList);
                }
                else
                {
                        /*
                         * Comportament in cazul aparitiei unei perturbatii
                         */
                        System.out.println("A disruption occured in the system.");
                        System.out.println("DF Search and Schedule processes are performed again.");
                }
        }
}
class searchServicesDFBehaviour extends OneShotBehaviour
{
        private OrderAgent orderAgent;
        private AID[] searchResult = new AID[10]; // resurse care efectueaza o operatie
        private int searchResultNo;
        private String[] OperationsList;


        public String[] getOperationsList() {
                return OperationsList;
        }
        public void setOperationsList(String[] extraOperations) {
                OperationsList = extraOperations;
        }
        searchServicesDFBehaviour(OrderAgent oa)
        {
                super(oa);
                orderAgent = oa;
        }
        public void action()
        {
```

```
DFAgentDescription dfAD = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
SearchConstraints ALL = new SearchConstraints();
ALL.setMaxResults(new Long(-1));
dfAD.addServices(sd);

String resourceList="";
searchResultNo = 0;

if(!orderAgent.isResourceBreakdown())
{
long startExecutionTime = System.currentTimeMillis();
System.out.println("Production of "+orderAgent.getOrderName()+" starts at: "+
TimeStampFormatterUtil.format(startExecutionTime));
orderAgent.setBeginExecutionTime(startExecutionTime);
}

orderAgent.setOrderState("DF search");
orderAgent.sendMessage(new
AID("Monitor",AID.ISLOCALNAME),orderAgent.getOrderState(),ACLMessage.INFORM,null);

for(int i=0;i<orderAgent.getCurrentOperationNo();i++)
{
        resourceList = "";
        searchResult = null;
        OperationsList = orderAgent.getCurrentOrderOperations();
        try
        {
                DFAgentDescription[] agentDescription =
DFService.search(this.myAgent, dfAD, ALL);

                searchResult=serviceSearch(agentDescription,OperationsList[i]);
                System.out.println("Resurse care efectueaza "+OperationsList[i]+": ");
                resourceList = "Available resources for "+OperationsList[i]+" operation:
\n";

                if(searchResult.length == 0)
                {
                        System.out.println("Nu exista nicio resursa valida pentru
operatia "+OperationsList[i]+".");
                        System.out.println("Productia este sistata.");
                }
                for(int j=0;j<searchResult.length;j++)
                {
                        if(searchResult[j]!=null)
                        {
                                searchResultNo++;
                                System.out.print("   "+searchResult[j].getLocalName());
                                resourceList += '\t'+searchResult[j].getLocalName()+'\n';
                        }
                }
                System.out.println("");
```

```
                                        orderAgent.getLogger().info(resourceList);
                                        for(int j=0;j<searchResult.length;j++)
                                        {
                                                if(searchResult[j]!=null)
                                                {
                                                        orderAgent.sendMessage(searchResult[j], null,
ACLMessage.QUERY_REF, OperationsList[i]);
                                                }
                                        }
                                }
                                catch(FIPAException fipaE)
                                {
                                        fipaE.printStackTrace();
                                        System.out.println("Serviciul nu a fost gasit.");
                                        orderAgent.getLogger().severe("Error searching the DFService!");
                                }
                        }
                        orderAgent.setSearchResourceNo(searchResultNo);
                }
                public AID[] serviceSearch(DFAgentDescription[] a, String service)
                {
                        AID [] resurse = new AID[10];
                        int k=0;
                        for (int i=0;i<a.length;i++)
                        {
                                DFAgentDescription dfd = new DFAgentDescription();
                                dfd = a[i];  //each agent in DF
                                AID agenţiD = dfd.getName(); // Agents AIDs.
                                Iterator it = dfd.getAllServices(); //Agent's service collection

                                while(it.hasNext())
                                {
                                        ServiceDescription sd = (ServiceDescription)it.next();
                                        if (sd.getName().equals(service))
                                        {
                                                resurse[k]=agenţiD;
                                                k++;
                                        }
                                }
                        }
                        return resurse;
                }
}
class UpdateKnowledgeBehaviour extends Behaviour
{
        private OrderAgent orderAgent;
        private boolean finishBehaviour = false;
        private int responseNo;
        private String[] chosenOperation;
        private String[] chosenResource;
        private int[] resourceLoad;
        private long startBehaviourTime;
```

```
	UpdateKnowledgeBehaviour(OrderAgent oa)
	{
		super(oa);
		orderAgent = oa;
	}
	public void onStart()
	{
		finishBehaviour = false;
		responseNo = 0;
		startBehaviourTime = System.currentTimeMillis();
		if(!orderAgent.isResourceBreakdown())
		{
			chosenOperation = new String[orderAgent.getCurrentOperationNo()];
			chosenResource = new String[orderAgent.getCurrentOperationNo()];
		}
		resourceLoad = new int[orderAgent.getOperationNo()];
		for(int i=0;i<resourceLoad.length;i++)
		{
			resourceLoad[i]=100;
		}
	}
	public void action()
	{
		ACLMessage receivedMessage =
this.myAgent.receive(MessageTemplate.MatchPerformative(ACLMessage.INFORM));
		if(receivedMessage!= null)
		{
			orderAgent.setOrderState("Scheduling");
			orderAgent.sendMessage(new
AID("Monitor",AID.ISLOCALNAME),orderAgent.getOrderState(),ACLMessage.INFORM,null);
			responseNo++;
			orderAgent.getLogger().info("Receiving INFORM message from "+
receivedMessage.getSender().getLocalName()+" with content: "+receivedMessage.getContent());

			System.out.println("INFORM
Expeditor:"+receivedMessage.getSender().getLocalName()+"  Continut:
"+receivedMessage.getContent()+"	Destinatar:"+this.myAgent.getLocalName());
			for(int i=0;i<orderAgent.getOrderOperations().length;i++)
			{

	if(orderAgent.getOrderOperations()[i].equals(receivedMessage.getConversationId()))
				{

	if(Integer.parseInt(receivedMessage.getContent())<resourceLoad[i])
					{
						resourceLoad[i] =
Integer.parseInt(receivedMessage.getContent());
						chosenResource[i] =
receivedMessage.getSender().getLocalName();
						chosenOperation[i] =
receivedMessage.getConversationId();
```

```
                                        }
                                }
                        }
                        if(responseNo == orderAgent.getSearchResourceNo() ||
(System.currentTimeMillis()-startBehaviourTime > 5000))
                        {
                                orderAgent.setChosenOperation(chosenOperation);
                                orderAgent.setChosenResource(chosenResource);

                                finishBehaviour = true;
                        }
                }
                else
                {
                        block();
                }
        }
        public boolean done()
        {
                return finishBehaviour;
        }
}
class sendProposeMessageBehaviour extends OneShotBehaviour
{
        private OrderAgent orderAgent;

        sendProposeMessageBehaviour(OrderAgent oa)
        {
                super(oa);
                orderAgent = oa;
        }
        public void action()
        {
                String operationsListGui="Schedule:\n";
                String schedule="";
                orderAgent.getLogger().info("The best schedule is:");

                for(int i=0;i<orderAgent.getChosenOperation().length;i++)
                {
                        System.out.println("Ordin: "+orderAgent.getOrderName()+" - Operatie:
"+orderAgent.getChosenOperation()[i]+" - Resursa_finala: "+orderAgent.getChosenResource()[i]);
                        schedule += '\t'+"Operation: "+orderAgent.getChosenOperation()[i]+" -
Resource: "+orderAgent.getChosenResource()[i]+'\n';
                        operationsListGui += orderAgent.getChosenOperation()[i]+" -
"+orderAgent.getChosenResource()[i]+'\n';
                }
                orderAgent.getLogger().info(schedule);
                orderAgent.sendMessage(new
AID("Monitor",AID.ISLOCALNAME),orderAgent.getOrderState(),ACLMessage.INFORM,operationsListGui)
;

                orderAgent.setOrderState("Break resource!");
```

```
                orderAgent.sendMessage(new
AID("Monitor",AID.ISLOCALNAME),orderAgent.getOrderState(),ACLMessage.INFORM,null);

                /*System.out.println("Waiting for resource to break down!");
                try
                {
                        Thread.sleep(3000);
                }
                catch(InterruptedException ie)
                {
                        ie.printStackTrace();
                }*/
                for(int i=0;i<orderAgent.getChosenOperation().length;i++)
                {
                        orderAgent.sendMessage(new
AID(orderAgent.getChosenResource()[i],AID.ISLOCALNAME),"",ACLMessage.PROPOSE,
orderAgent.getChosenOperation()[i]);
                }
        }
}
class receiveAcceptMessageBehaviour extends Behaviour
{
        private OrderAgent orderAgent;
        private int acceptMessages;
        private boolean finishBehaviour = false;
        private long startBehaviourTime;

        receiveAcceptMessageBehaviour(OrderAgent oa)
        {
                super(oa);
                orderAgent = oa;
        }
        public void onStart()
        {
                acceptMessages = 0;
                finishBehaviour = false;
                startBehaviourTime = System.currentTimeMillis();
                if(!orderAgent.isResourceBreakdown())
                {
                        for(int i=0;i<orderAgent.getOrderOperations().length;i++)
                        {
                                orderAgent.arrayList.add(i, null);
                        }
                }
        }
        public void action()
        {
                ACLMessage receivedMessage =
this.myAgent.receive(MessageTemplate.MatchPerformative(ACLMessage.ACCEPT_PROPOSAL));
                if(receivedMessage!= null)
                {
                        String resource,operation;
```

```
                        acceptMessages++;
                        System.out.println("ACCEPT_PROPOSAL
Expeditor:"+receivedMessage.getSender().getLocalName()+"
Destinatar:"+this.myAgent.getLocalName());
                        orderAgent.getLogger().info("Receiving ACCEPT_PROPOSAL message from "+
receivedMessage.getSender().getLocalName());
                        resource = receivedMessage.getSender().getLocalName();
                        operation = receivedMessage.getContent();
                        for(int i=0;i<orderAgent.getOrderOperations().length;i++)
                        {
                                if(operation.equals(orderAgent.getOrderOperations()[i]))
                                {
                                        orderAgent.arrayList.set(i, new
ResourceOperationList(operation,resource));
                                }
                        }
                }
                else
                {
                        if(acceptMessages == orderAgent.getOperationNo())
                        {
                                System.out.println("Am primit mesaj ACCEPT_PROPOSAL de la toate!");

                                finishBehaviour = true;
                        }
                        else if (System.currentTimeMillis() - startBehaviourTime > 5000)
                        {
                                System.out.println("Nu am primit mesaj de la toate resursele!");
                                orderAgent.setResourceBreakdown(true);
                                int operatii_nealocate = (orderAgent.getOperationNo()-
acceptMessages);
                                System.out.println("Nr de operatii nealocate: "+operatii_nealocate);
                                String [] operatiiNealocate = new String[operatii_nealocate];
                                int k=0;
                                for(int i=0;i<orderAgent.arrayList.size();i++)
                                {
                                        if(orderAgent.arrayList.get(i)==null)
                                        {
                                                operatiiNealocate[k] =
orderAgent.getOrderOperations()[i];
                                                k++;
                                        }
                                }
                                for(int i=0;i<operatii_nealocate;i++)
                                {
                                        System.out.println("Operatii nealocate: "+operatiiNealocate[i]);
                                }
                                orderAgent.setCurrentOperationNo(operatii_nealocate);
                                orderAgent.setCurrentOrderOperations(operatiiNealocate);
                                orderAgent.seqBehaviour.reset();
                                finishBehaviour = true;
                        }
```

```
		}
	}
	public boolean done()
	{
		return finishBehaviour;
	}
}
class PalletCellEntryBehaviour extends OneShotBehaviour
{
	private String orderLocationTCPMessage;
	private OrderAgent orderAgent;

	PalletCellEntryBehaviour(OrderAgent oa)
	{
		super(oa);
		orderAgent = oa;
	}
	public void onStart()
	{
		System.out.println("Plasati paleta pe conveior...");
		orderLocationTCPMessage = "000" +
orderAgent.getOrderName().substring(orderAgent.getOrderName().length()-1) + "0106";
	}
	public void action()
	{
		String receivedMessage="";
		boolean palet_in_cell = false;
		orderAgent.setOrderState("Plasati paleta!!");
		orderAgent.sendMessage(new
AID("Monitor",AID.ISLOCALNAME),orderAgent.getOrderState(),ACLMessage.INFORM,null);

		while(!palet_in_cell)
		{
		  try
		  {
			orderAgent.getClientSocket().sendMessage(orderLocationTCPMessage); //
mesaj tip 1
		  }
		  catch(IOException ioe)
		  {
			ioe.printStackTrace();
		  }
		  receivedMessage = orderAgent.getClientSocket().readSocket();
		  orderAgent.getClientSocket().parseMessage(receivedMessage);
		  palet_in_cell = orderAgent.getClientSocket().cellEntry(orderAgent.getOrderName());
		}
	}
}
class productionExecutionBehaviour extends OneShotBehaviour
{
	private OrderAgent orderAgent;
	private boolean post = false;
```

```
	private String orderLocationTCPMessage = "";// type 1 message
	private String operationDoneTCPMessage = "";// type 3 message
	private String nextOperation, nextResource, lastResource = "";
	protected ArrayList<ResourceOperationList> arrayList1;
	private String resourceListTCPMessage = "";// type 2 message - 54 characters

	productionExecutionBehaviour(OrderAgent oa)
	{
		super(oa);
		orderAgent = oa;
	}
	public void onStart()
	{
		for(int i=0;i<orderAgent.arrayList.size();i++)
		{
			orderAgent.arrayList.get(i).printListItem();
		}
		arrayList1 = orderAgent.getArrayList();
		orderLocationTCPMessage = "000" +
orderAgent.getOrderName().substring(orderAgent.getOrderName().length()-1) + "0106";
		operationDoneTCPMessage = "000" +
orderAgent.getOrderName().substring(orderAgent.getOrderName().length()-1) + "0308";
		resourceListTCPMessage = "000" +
orderAgent.getOrderName().substring(orderAgent.getOrderName().length()-1) + "0254";

		for(int i=0;i<orderAgent.arrayList.size();i++)
		{
			resourceListTCPMessage +=
"00"+orderAgent.arrayList.get(i).getListResourceItem().charAt(orderAgent.arrayList.get(i).getListResourc
eItem().length()-1);
		}
		resourceListTCPMessage += "255";
		while(resourceListTCPMessage.length()<56)
		{
			resourceListTCPMessage+="0";
		}
		try
	  {
		 orderAgent.getClientSocket().sendMessage(resourceListTCPMessage);
	  }
	  catch(IOException ioe)
	  {
		ioe.printStackTrace();
	  }
		orderAgent.getClientSocket().readSocket();

	}
	public void action()
	{
		post = false;
		if(arrayList1.size()>0)
		{
```

```
                        orderAgent.getLogger().info(arrayList1.size()+" operations left!");
                        nextOperation = arrayList1.get(0).getListOperationItem();
                        nextResource = arrayList1.get(0).getListResourceItem();

                        System.out.println("Next operation:"+nextOperation);
                        orderAgent.getLogger().info("Next operation:"+nextOperation+" will be
executed by resource:"+nextResource);

                        if(lastResource.equals(nextResource))
                        {
                                System.out.println("Next operation is on the same resource as last one.
No transport needed.");
                                orderAgent.getLogger().info("Next operation is on the same resource as
last one. No transport needed.");
                        }
                        else
                        {
                                if(!lastResource.equals(""))
                                {
                                        robotDone(operationDoneTCPMessage +
Integer.parseInt(lastResource.substring(lastResource.length()-1)));
                                }
                                System.out.println("Transporting "+orderAgent.getOrderName()+" to
"+nextResource+"...");
                                orderAgent.getLogger().info("Transporting
"+orderAgent.getOrderName()+" to "+nextResource+"...");
                                orderAgent.setOrderState("Transport to - "+nextResource);
                                orderAgent.sendMessage(new
AID("Monitor",AID.ISLOCALNAME),orderAgent.getOrderState(),ACLMessage.INFORM,null);
                                //transport
                                while(post==false)
                                {
                                        String receivedMessage="";
                                   try
                                   {

        orderAgent.getClientSocket().sendMessage(orderLocationTCPMessage); // mesaj tip 1
                                   }
                                   catch(IOException ioe)
                                   {
                                        ioe.printStackTrace();
                                   }
                                   receivedMessage = orderAgent.getClientSocket().readSocket();
                                   try
                                   {
                                        Thread.sleep(500);
                                   }
                                   catch(InterruptedException ie)
                                   {
                                        ie.printStackTrace();
                                   }
                                   orderAgent.getClientSocket().parseMessage(receivedMessage);
```

```
                                        post =
orderAgent.getClientSocket().arriveAtWorkstation(orderAgent.getOrderName(), nextResource);
                                    }
                            }
                            System.out.println("DONE TRANSPORTING...");
                            orderAgent.sendMessage(new
AID(nextResource,AID.ISLOCALNAME),"",ACLMessage.REQUEST, nextOperation);
                            orderAgent.setOrderState("Execution - "+nextOperation);
                            orderAgent.sendMessage(new
AID("Monitor",AID.ISLOCALNAME),orderAgent.getOrderState(),ACLMessage.INFORM,null);
                            lastResource = arrayList1.get(0).getListResourceItem();
                            arrayList1.remove(0);
                            Behaviour receiveConfirmMessage = new
receiveConfirmMessageBehaviour(this,orderAgent, nextOperation);
                            myAgent.addBehaviour(receiveConfirmMessage);
                    }
                    else
                    {
                            long endExecutionTime = System.currentTimeMillis();
                            long beginExecTime = orderAgent.getBeginExecutionTime();
                            orderAgent.setOrderState("END of production");
                            orderAgent.sendMessage(new
AID("Monitor",AID.ISLOCALNAME),orderAgent.getOrderState(),ACLMessage.INFORM,"");

            robotDone(operationDoneTCPMessage+Integer.parseInt(lastResource.substring(lastResource.le
ngth()-1)));
                            System.out.println("END of production for "+orderAgent.getOrderName()+" at
"+TimeStampFormatterUtil.format(endExecutionTime));
                            orderAgent.getLogger().info("END of production for
"+orderAgent.getOrderName()+".");

                            long timeElapse = endExecutionTime - beginExecTime;
                            System.out.println("Execution time for "+orderAgent.getLocalName()+":
"+TimeStampFormatterUtil.format(timeElapse));
                            orderAgent.getClientSocket().closeSocket();
                    }

            }
            public void robotDone(String operationDoneTCPMessage)
            {
                    try
              {
                            orderAgent.getClientSocket().sendMessage(operationDoneTCPMessage+"1"); //
mesaj tip 3
              }
              catch(IOException ioe)
              {
                    ioe.printStackTrace();
              }
                    orderAgent.getClientSocket().readSocket();
                    try
              {
```

```
                Thread.sleep(1000);
            }
            catch(InterruptedException ioe)
            {
                ioe.printStackTrace();
            }
                try
            {
                    orderAgent.getClientSocket().sendMessage(operationDoneTCPMessage+"0"); //
mesaj tip 3
            }
            catch(IOException ioe)
            {
                ioe.printStackTrace();
            }
                orderAgent.getClientSocket().readSocket();
        }
}
class receiveConfirmMessageBehaviour extends Behaviour
{
        private OrderAgent orderAgent;
        private boolean finishBehaviour = false;
        private String currentOperation = null;
        private productionExecutionBehaviour beh;

        receiveConfirmMessageBehaviour(productionExecutionBehaviour b, OrderAgent oa, String
operation)
        {
                super(oa);
                orderAgent = oa;
                currentOperation = operation;
                beh = b;
        }
        public void onStart()
        {
                finishBehaviour = false;
        }
        public void action()
        {
                ACLMessage receivedMessage =
this.myAgent.receive(MessageTemplate.MatchPerformative(ACLMessage.CONFIRM));
                if(receivedMessage!= null)
                {
                        if(receivedMessage.getConversationId().equals(currentOperation))
                        {
                                System.out.println("CONFIRM
Expeditor:"+receivedMessage.getSender().getLocalName()+"  ConvID:
"+receivedMessage.getConversationId()+"         Destinatar:"+this.myAgent.getLocalName());
                                orderAgent.getLogger().info("Receiving CONFIRM message from "+
receivedMessage.getSender().getLocalName()+"for operation "+receivedMessage.getConversationId());
                                beh.action();
                                finishBehaviour = true;
```

```
                        }
                }
                else block();
        }
        public boolean done()
        {
                return finishBehaviour;
        }
}
```

Agentul de tip Ordin – cazul urmatoarea operatie

```
package OrderAgentNextOperation;

import java.util.logging.*;
import java.io.*;
import java.util.*;

import jade.core.Agent;
import jade.core.AID;
import jade.core.behaviours.*;
import jade.util.leap.Iterator;
import jade.domain.FIPAException;
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.*;
import jade.lang.acl.ACLMessage;

public class OrderNextOperation extends Agent
{
        private Logger logger = Logger.getLogger("MyLog");
        private AID[] searchResult = new AID[10]; // resurse care efectueaza o operatie
        private int searchResultNo;
        private int responseNo = 0;
        private String resourceList="";
        private String operationsListLogger="List of operations:\n";
        private int resourceLoad;
        private ArrayList<String> operationsList;
        private String nextOperation,nextResource, lastResource = "";
        private boolean executionReady = false;
        private Behaviour b1,b2;
        private int k, operationNo = 0;
        private RandomAccessFile randomAccesFile;
        private TCPClient clientSocket;
        private String orderState="";
        private boolean post = false;
private String orderLocationTCPMessage,resourceListTCPMessage,operationDoneTCPMessage="";
        private String filePath;
        private String blankMessage = "";

        protected void setup()
        {
                String startMessage;
```

```
                System.out.println("Hello. My name is "+ getLocalName());

                orderState = "Initialization";
                sendMessage(new
AID("Monitor",AID.ISLOCALNAME),orderState,ACLMessage.INFORM,null);
                startMessage = "000" + getLocalName().substring(getLocalName().length()-1);
                orderLocationTCPMessage = startMessage + "0106";
                resourceListTCPMessage  = startMessage + "0254";
                operationDoneTCPMessage = startMessage + "0308";
            FileHandler fileHandler;
            clientSocket = new TCPClient();

            try
            {
                    clientSocket.startClientSocket(this.getLocalName());
            }
            catch(IOException ioe)
            {
                    ioe.printStackTrace();
            }
            try
            {
               filePath = "/root/javadown/pachet/Logger/"+this.getLocalName()+"Logger.log";
               fileHandler = new FileHandler(filePath, false); // append = true
               logger.setUseParentHandlers(false); // no console output
               logger.addHandler(fileHandler);
               SimpleFormatter formatter = new SimpleFormatter();
               fileHandler.setFormatter(formatter);
            } catch (Exception e)
            {
               e.printStackTrace();
            }
            logger.info("Order "+this.getLocalName()+" launched!");

                    operationsList = new ArrayList<String>();

                    readOperationsFile();//citire lista operatii din fisier
                    logger.info("The order has "+this.operationNo+" operations.");
                    logger.info(operationsListLogger);

                    b1 = new nextOperationBehaviour(this);
                    this.addBehaviour(b1);
                    b2 = new receiveMessageBehaviour(this);
                    this.addBehaviour(b2);
        }
        class nextOperationBehaviour extends TickerBehaviour
        {
                    nextOperationBehaviour(Agent a)
                    {
                            super(a,100);
                    }
                    public void onTick()
```

```
		{
			if(k==0)
			{
				resourceLoad = 100;
				nextOperation = operationsList.get(0);
				System.out.println("First operation:"+nextOperation);
				System.out.println("Size: "+operationsList.size());
				DFSearch(nextOperation);
				k++;
			}
			else if(executionReady==true)
			{
				if(operationsList.size()==1)
				{
					orderState = "END of production";
					sendMessage(new
AID("Monitor",AID.ISLOCALNAME),orderState,ACLMessage.INFORM,"");

	robotDone(operationDoneTCPMessage+Integer.parseInt(lastResource.substring(lastResource.le
ngth()-1)));
					System.out.println("END of production for
"+this.myAgent.getLocalName()+".");
					logger.info("END of production for
"+this.myAgent.getLocalName()+".");
					clientSocket.closeSocket();
					this.stop();
					this.myAgent.doDelete();
				}
				else
				{
					operationsList.remove(0);
					resourceLoad = 100;
					responseNo = 0;
					nextOperation = operationsList.get(0);
					System.out.println("Next operation:"+nextOperation);
					//System.out.println("Size: "+operationsList.size());
					DFSearch(nextOperation);
					executionReady = false;
				}
			}
		}
	}
	class receiveMessageBehaviour extends CyclicBehaviour
	{
		receiveMessageBehaviour(Agent a)
		{
			super(a);
		}
		public void action()
		{
			ACLMessage receivedMessage = receive();
			if(receivedMessage!= null)
```

```
{
	switch(receivedMessage.getPerformative())
	{
		case(ACLMessage.INFORM):
		{
			orderState = "Scheduling";
			sendMessage(new
AID("Monitor",AID.ISLOCALNAME),orderState,ACLMessage.INFORM,null);
			responseNo++;
			logger.info("Receiving INFORM message from "+
receivedMessage.getSender().getLocalName());
			logger.info("Message content: "+
receivedMessage.getContent());

			System.out.println("INFORM
Expeditor:"+receivedMessage.getSender().getLocalName()+"  Mesaj: "+receivedMessage.getContent()+"
	Destinatar:"+this.myAgent.getLocalName());

	if(nextOperation.equals(receivedMessage.getConversationId()))
			{

	if(Integer.parseInt(receivedMessage.getContent())<resourceLoad)
				{
					resourceLoad =
Integer.parseInt(receivedMessage.getContent());
					nextResource =
receivedMessage.getSender().getLocalName();
				}
			}
			if(responseNo == searchResultNo)
			{
				sendMessage(new
AID(nextResource,AID.ISLOCALNAME),"",ACLMessage.PROPOSE, nextOperation);
			}
		}break;
		case(ACLMessage.ACCEPT_PROPOSAL):
		{
			System.out.println("ACCEPT_PROPOSAL
Expeditor:"+receivedMessage.getSender().getLocalName()+"  ConvID:
"+receivedMessage.getConversationId());
			logger.info("Receiving ACCEPT_PROPOSAL message
from "+ receivedMessage.getSender().getLocalName());
			resourceListTCPMessage +=
"00"+nextResource.charAt(nextResource.length()-1);

			//System.out.println("Size: "+operationsList.size());
			if(operationsList.size()==1)
			{
				resourceListTCPMessage += "255";
			}
			int dimensiune = 56 - resourceListTCPMessage.length();
			blankMessage = "";
```

```
                                        for(int i=0;i<dimensiune;i++)
                                        {
                                                blankMessage +="0";
                                        }

                                        try
                                    {

clientSocket.sendMessage(resourceListTCPMessage+blankMessage);
                                         //System.out.println(resourceListTCPMessage);
                                    }
                                    catch(IOException ioe)
                                    {
                                        ioe.printStackTrace();
                                    }
                                        clientSocket.readSocket();
                                        startExecution();
                                }break;
                                case(ACLMessage.CONFIRM):
                                {
                                        System.out.println("CONFIRM
Expeditor:"+receivedMessage.getSender().getLocalName()+"  ConvID:
"+receivedMessage.getConversationId()+"         Destinatar:"+this.myAgent.getLocalName());
                                        logger.info("Receiving CONFIRM message from "+
receivedMessage.getSender().getLocalName()+"for operation "+receivedMessage.getConversationId());
                                        //resursa a terminat executia
                                        executionReady = true;
                                }break;
                                default:
                                {
                                        System.out.println("Unknown message from
"+receivedMessage.getSender().getLocalName());
                                        logger.warning("Received unknown message!");
                                }
                            }
                        }
                        else
                        {
                                block();
                        }
                }
        }


        public void DFSearch(String op)
        {
                resourceList = "";
                orderState = "DF search";
                sendMessage(new
AID("Monitor",AID.ISLOCALNAME),orderState,ACLMessage.INFORM,null);
                searchResultNo = 0;
                DFAgentDescription dfAD = new DFAgentDescription();
```

```
		ServiceDescription sd = new ServiceDescription();
		SearchConstraints ALL = new SearchConstraints();
		ALL.setMaxResults(new Long(-1));
		dfAD.addServices(sd);

		try
		{
			DFAgentDescription[] agentDescription = DFService.search(this, dfAD, ALL);
			searchResult=serviceSearch(agentDescription,op);

			System.out.println("Resurse care efectueaza "+op+": ");
			resourceList = "Available resources for "+op+" operation: \n";
			for(int j=0;j<searchResult.length;j++)
			{
				if(searchResult[j]!=null)
				{
					searchResultNo++;
					System.out.print("   "+searchResult[j].getLocalName());
					resourceList += '\t'+searchResult[j].getLocalName()+'\n';
				}
			}
			System.out.println("");
			logger.info(resourceList);
			for(int j=0;j<searchResult.length;j++)
			{
				if(searchResult[j]!=null)
				{
					sendMessage(searchResult[j], null, ACLMessage.QUERY_REF,
op);

				}
			}
		}
		catch(FIPAException fipaE)
		{
			fipaE.printStackTrace();
			System.out.println("Serviciul nu a fost gasit.");
			logger.severe("Error searching the DFService!");
		}
	}

	public AID[] serviceSearch(DFAgentDescription[] a, String service)
	{
		AID [] resurse = new AID[10];
		int k=0;
		for (int i=0;i<a.length;i++)
		{
			DFAgentDescription dfd = new DFAgentDescription();
			dfd = a[i];  //each agent in DF
			AID agentiD = dfd.getName(); // Agents AIDs.
			Iterator it = dfd.getAllServices(); //Agent's service collection

			while(it.hasNext())
```

```
                        {
                                ServiceDescription sd = (ServiceDescription)it.next();
                                if (sd.getName().equals(service))
                                {
                                        resurse[k]=agenţiD;
                                        k++;
                                }
                        }
                }
                return resurse;
        }

        public void sendMessage(AID a,String content, int message_type, String covID)
        {
                ACLMessage mesajTrimis;
                mesajTrimis = new ACLMessage(message_type);
                mesajTrimis.addReceiver(a);
                mesajTrimis.setContent(content);
                mesajTrimis.setConversationId(covID);
                this.send(mesajTrimis);
                logger.info("Sending "+ACLMessage.getPerformative(message_type)+" message to
"+a.getLocalName()+".");
        }

        public void startExecution()
        {
                post = false;
                logger.info("Next operation:"+nextOperation+" will be executed by
resource:"+nextResource);
                //sendMessage(new
AID("Monitor",AID.ISLOCALNAME),orderState,ACLMessage.INFORM,);
                if(lastResource.equals(nextResource))
                {
                        System.out.println("Next operation is on the same resource as last one. No
transport needed.");
                        logger.info("Next operation is on the same resource as last one. No transport
needed.");
                        try
                  {
                        Thread.sleep(500);
                  }
                  catch(InterruptedException ie)
                  {
                        ie.printStackTrace();
                  }
                }
                else
                {
                        if(!lastResource.equals(""))
                        {
                                robotDone(operationDoneTCPMessage +
Integer.parseInt(lastResource.substring(lastResource.length()-1)));
```

```
                        }
                        System.out.println("Transporting "+this.getLocalName()+" to
"+nextResource+"...");
                        logger.info("Transporting "+this.getLocalName()+" to "+nextResource+"...");
                        orderState = "Transport to - "+nextResource;
                        sendMessage(new
AID("Monitor",AID.ISLOCALNAME),orderState,ACLMessage.INFORM,nextOperation+" -
"+nextResource);
                        //transport
                        while(post==false)
                        {
                                //System.out.println("Transporting to "+nextResource+"....");
                                String receivedMessage="";
                          try
                          {
                                clientSocket.sendMessage(orderLocationTCPMessage); // mesaj tip 1
                          }
                          catch(IOException ioe)
                          {
                                ioe.printStackTrace();
                          }
                          receivedMessage = clientSocket.readSocket();
                          try
                          {
                                Thread.sleep(1000);
                          }
                          catch(InterruptedException ie)
                          {
                                ie.printStackTrace();
                          }
                          clientSocket.parseMessage(receivedMessage);
                                post = clientSocket.arriveAtWorkstation(this.getLocalName(),
nextResource);
                        }
                    try
                    {
                        Thread.sleep(1000);
                    }
                    catch(InterruptedException ie)
                    {
                        ie.printStackTrace();
                    }
                }
                System.out.println("DONE TRANSPORTING...");
                sendMessage(new AID(nextResource,AID.ISLOCALNAME),"",ACLMessage.REQUEST,
nextOperation);
                orderState = "Execution - "+nextOperation;
                sendMessage(new
AID("Monitor",AID.ISLOCALNAME),orderState,ACLMessage.INFORM,null);
                lastResource = nextResource;
                //System.out.println("last resource:"+lastResource);
        }
```

```
	public void readOperationsFile()
	{
		try
		{
			randomAccesFile = new
RandomAccessFile("/root/javadown/pachet/OrderAgentNextOperation/Lista"+this.getLocalName()+".tx
t", "r");
		}
		catch(FileNotFoundException fnf)
		{
			fnf.printStackTrace();
			System.exit(0);
		}
		FileInputStream fileStream = null;
		String readLineString;
		try {
			operationNo = Integer.parseInt(randomAccesFile.readLine());
			while ((readLineString = randomAccesFile.readLine()) != null)
			{
				System.out.println("" +readLineString);
				operationsList.add(readLineString);
				operationsListLogger += '\t'+readLineString+'\n';
			}
			randomAccesFile.seek(randomAccesFile.getFilePointer());
		}
		catch (IOException e)
		{
			e.printStackTrace();
		} finally
		{
			try
			{
				if (fileStream != null)
					fileStream.close();}
			catch (IOException ex)
			{
				ex.printStackTrace();}
		}
	}
	public void robotDone(String operationDoneTCPMessage)
	{
		try
	  {
		clientSocket.sendMessage(operationDoneTCPMessage+"1"); // mesaj tip 3   }
	  catch(IOException ioe)
	  {
		ioe.printStackTrace();
	  }
		clientSocket.readSocket();
		try
	  {
		Thread.sleep(3000);
```

```
            }
            catch(InterruptedException ioe)
            {
                  ioe.printStackTrace();
            }
                  try
            {
                  clientSocket.sendMessage(operationDoneTCPMessage+"0"); // mesaj tip 3
            }
            catch(IOException ioe)
            {
                  ioe.printStackTrace();
            }
                  clientSocket.readSocket();
          }
}
```

Agentul de tip Resursa

```
package ResourceAgent;
import jade.core.*;
import java.util.logging.*;
import jade.core.behaviours.*;
import jade.domain.FIPAException;
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.*;
import jade.gui.*;
import jade.lang.acl.ACLMessage;

public class ResourceAgent extends GuiAgent
{
private DFAgentDescription dfd;
private String[] Operations = new String[2];
private String pathFile = "";
private String operationsList="List of operations:\n";
private String fileName = "";
public ResourceInformation resourceInformation;
private Logger logger;
private agentGui gui;
static final int QUIT = 0;
private int command;
protected void setup(){
FileHandler fileHandler;
                  System.out.println("Hello. My name is "+ getLocalName());
                  logger = Logger.getLogger(this.getLocalName());
            try
            {
              fileName = this.getLocalName()+"Logger.log";
                  this.setPathFile("E:/Roxana/Logger/"+fileName);
              fileHandler = new FileHandler(pathFile, false); // append = true
              logger.setUseParentHandlers(false); // no console output
              logger.addHandler(fileHandler);
              SimpleFormatter formatter = new SimpleFormatter();
```

```
            fileHandler.setFormatter(formatter);
        } catch (Exception e)
        {
            e.printStackTrace();
        }
        logger.info("Resource "+this.getLocalName()+" launched!");
                dfd = new DFAgentDescription();
                resourceInformation = new ResourceInformation(this.getLocalName());
                dfd.setName(getAID());

                Operations = resourceInformation.getOperations(this.getLocalName());
                logger.info("Operations executed by the resource:");

                gui = new agentGui();
        gui.startGUI(this);

                for(int i=0;i<Operations.length;i++)
                {
                        operationsList += '\t'+Operations[i]+'\n';
                        ServiceDescription sd = new ServiceDescription();
                        sd.setType(Operations[i]);
                        sd.setName(Operations[i]);
                        dfd.addServices(sd);
                }
                dfRegister(this,dfd);
                logger.info(operationsList);
                gui.readLogFile();

                addBehaviour(new CyclicBehaviour(this)
                {
                        public void action()
                        {
                                ACLMessage receivedMessage = receive();
                                if(receivedMessage!= null)
                                {
                                        switch(receivedMessage.getPerformative())
                                        {
                                                case(ACLMessage.QUERY_REF):
                                                {
                                                        System.out.println("***  Message type
QUERY_REF");
                                                        System.out.println("Destinatar:
"+getLocalName()+"      Expeditor: "+receivedMessage.getSender().getLocalName()+"  Content:
"+receivedMessage.getContent());
                                                        logger.info("Receiving QUERY_REF message
from "+ receivedMessage.getSender().getLocalName());
                                                        gui.readLogFile();

                                                        resourceInformation.setResourceState("IDLE");
                                                        gui.resetStateLabel();
                                                        gui.resetImagePicture();
```

```
                                                ACLMessage sentMessage = new
ACLMessage(ACLMessage.INFORM);
        sentMessage.addReceiver(receivedMessage.getSender());
                        sentMessage.setContent(""+resourceInformation.getResourceLoad());
                                sentMessage.setConversationId(receivedMessage.getConversationId());
                                                send(sentMessage);
                                                logger.info("Send INFORM message to
"+receivedMessage.getSender().getLocalName());
                                                gui.readLogFile();
                                        }break;
                                        case(ACLMessage.PROPOSE):
                                        {
                                                System.out.println("***  Message type
PROPOSE!");
                                                System.out.println("Destinatar:
"+getLocalName()+"      Expeditor: "+receivedMessage.getSender().getLocalName()+"  Continut:
"+receivedMessage.getContent());
                                                logger.info("Receiving PROPOSE message from
"+ receivedMessage.getSender().getLocalName()+" for operation "+ receivedMessage.getContent());
                                                gui.readLogFile();

                                                ACLMessage sentMessage = new
ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
        sentMessage.addReceiver(receivedMessage.getSender());
                                                sentMessage.setContent(null);
                                                send(sentMessage);
                                                logger.info("Send ACCEPT_PROPOSAL message
to "+receivedMessage.getSender().getLocalName());
                                                gui.readLogFile();
resourceInformation.setResourceLoad(resourceInformation.getResourceLoad()+1);
                                                gui.resetStateLabel();
                                                System.out.println("Incarcarea pt resursa
"+this.myAgent.getLocalName()+" este "+resourceInformation.getResourceLoad());
                                                }break;
                                        case(ACLMessage.REQUEST):                       {
                                                System.out.println("***  Message type
REQUEST!");
                                                System.out.println("Destinatar:
"+getLocalName()+"      Expeditor: "+receivedMessage.getSender().getLocalName()+"  Continut:
"+receivedMessage.getContent());
                                                logger.info("Receiving REQUEST message from
"+ receivedMessage.getSender().getLocalName()+" for operation "+ receivedMessage.getContent());
                                                gui.readLogFile();


        resourceInformation.setResourceState("WORK_IN_PROGRESS");

        resourceInformation.setCurrentOrder(receivedMessage.getSender().getLocalName());

        resourceInformation.setCurrentOperation(receivedMessage.getConversationId());
                                                gui.resetStateLabel();
                                                gui.resetImagePicture();
```

```
                                                    logger.info("Executing operation
"+receivedMessage.getConversationId()+" on "+receivedMessage.getSender().getLocalName()+"...");
                                                    System.out.println("Executing operation
"+receivedMessage.getConversationId()+" on "+receivedMessage.getSender().getLocalName()+"...");
                                                    gui.readLogFile();
                                                    gui.startCounter();
                                                    //executie operatie
                                                    try
                                                    {
                                                            Thread.sleep(5000);
                                                    }
                                                    catch(Exception e)
                                                    {
                                                            e.printStackTrace();
                                                    }

                                                    ACLMessage sentMessage = new
ACLMessage(ACLMessage.CONFIRM);

        sentMessage.addReceiver(receivedMessage.getSender());
                                                    sentMessage.setContent(null);

        sentMessage.setConversationId(receivedMessage.getConversationId());
                                                    send(sentMessage);


        resourceInformation.setResourceLoad(resourceInformation.getResourceLoad()-1);
resourceInformation.setCurrentOrder("");
resourceInformation.setCurrentOperation("");
resourceInformation.setResourceState("IDLE");
gui.resetStateLabel();
gui.resetImagePicture();
logger.info("Send CONFIRM message to "+receivedMessage.getSender().getLocalName());
gui.readLogFile();
}break;
default:{
logger.warning("Message received from "+receivedMessage.getSender().getLocalName()+" not
understood!");
                                                    gui.readLogFile();}}}
                                    else
                                    {
                                            block(); }}}});}
        public void dfRegister(Agent a, DFAgentDescription dfd)
        {
                try
                {
                        DFService.register(a, dfd);

                        logger.info("Successfully register to DF!");
                        gui.readLogFile();}
                catch(FIPAException fe)
```

```
        {
                fe.printStackTrace();
                System.out.println("Error !");
                logger.severe("Error while registering the agent to DF!");
                gui.readLogFile();
                System.exit(0);}}}
public void setPathFile(String a)
{
        this.pathFile=a;
}
public String getPathFile()
{
        return this.pathFile;
}}
```

Bibliografie

- Bussmann, S., & McFarlane, D. (1999). Rationales for Holonic Control Systems, Proceedings of *IMS99*(Leuven, Belgium).

- Dilts, D.M., Boyd, N.P. and Whorms, H.H. (1991).The Evolution of Control Architectures for Automated Manufacturing Systems. In *Journal of Manufacturing Systems*, Vol. 10, Issue 1, 79–93.

- Meyer, G. G., Främling, K., Holmström, J.(2009). Intelligent Products: a survey.In *Computers in Industry*, 60, 137-148.

- Meyer, G. G.,& Wortmann, H. (2010). Robust planning and control using intelligent product. In W. Aalst, J. Mylopoulos, N. M. Sadeh, M. J. Shaw, C. Szyperski, & E. David, (Eds.), *Agent-mediated electronic commerce. Designing trading strategies and mechanisms for electronic markets*. Lecture notes in business information processing, Springer, Vol. 59, 163-177.

- McFarlane, D.,Sarma, S., Chirn, J. L., Wong, C. Y., Ashton, K.(2002). The Intelligent Product in Manufacturing Control.In *Journal of EAIA.*

- McFarlane, D., Parlikad, A., Neely, A., Thorne, A. (2013). A framework for Distributed Intelligent Automation System Developments. *In Service Orientation in Holonic and Multi Agent Manufacturing and Robotics, Studies in Computational Intelligence,*Vol. 472, 2013, 313-326.

- McFarlane, D., Giannikas, V., Wong, C.Y., Harrison, M.(2013). Product intelligence in industrial control:Theory and practice. In *Annual Reviews in Control* 37, 69-88.

- Trentesaux, D., & Thomas, A. (2012). Product-Driven control: a State of the Art and Future Trends. In INCOM 2012: *14th IFAC Symposium on Information Control Problems in Manufacturing*. Bucharest.

- Trentesaux, D., & Thomas, A. (2013).Product-Driven Control: Concept, Literature Review and Future Trends. In *Service Orientation in Holonic and Multi Agent Manufacturing and Robotics*,*Studies in Computational Intelligence* Volume 472, 2013, pp 135-150.

- Van Brussel, H., Wyns, J., Valckenaers, P., Bongaerts, L., & Peeters, P. (1998). Reference Architecture for Holonic Manufacturing Systems: PROSA. In *Computers In Industry*, Vol. 37, No. 3, 255 – 276.

- Wong, C.Y. , McFarlane, D., Zaharudin, A., Agarwal, V.(2002). The Intelligent Product Driven Supply Chain. In 2002 *IEEE International Conference on Systems, Man and Cybernetics.*

- Bellifemine F., Caire G., GreenwoodD. (2007). Developing Multi-Agent Systems with JADE. John Wiley & Sons.

- Tutorial JADE http://www.iro.umontreal.ca/~vaucher/Agents/Jade/JadePrimer.html

- Configurare retea wireless modul Overo http://wiki.gumstix.org/index.php?title=Overo_Wifi