Multi-Agent Control System

Atef Gharbi[1], Hamza Gharsellaoui[2] and Samir Ben Ahmed[3]
[1]*Department of Computer Sciences, INSAT, Tunis, Tunisia*
[2]*Department of Computer Sciences, ESTI, Tunis, Tunisia*
[3]*Department of Computer Sciences, FST, Tunis, Tunisia*

Keywords: Multi-agent System, Distributed Planning, JADE Platform.

Abstract: The paper deals with distributed planning in a Multi-Agent System (MAS) constituted by several intelligent agents each one has to interact with the other autonomous agents. The problem faced is how to ensure a distributed planning through the cooperation in our multi-agent system. Finally, we use JADE platform to create agents and ensure the communication between them.
A Benchmark Production System is used as a running example to explain our contribution.

1 INTRODUCTION

While Multi-Agent System (MAS) is a concept mainly used in research (xia Dou et al., 2014), by adapting it we must face various problems, some of which are serious enough to place the utility of MAS in the doubt. Since we wish to use the MAS in large scales, concurrent systems, and since we wish to address not very frequent, but demanding problems (Liu et al., 2014), MAS can become arbitrarily complex if MAS can not provide guarantees which help to order the system and ensure the progression of the total application. We can not pretend the unicity nor the exactitude of an agent definition, however the most adapted one presented by (Ferber,) where an agent is defined as a physical or virtual entity (i) which is capable of acting in an environment; (ii) which can communicate directly with other agents; (iii) which is driven by a set of tendencies (in the form of individual objectives or of a satisfaction/survival function which it tries to optimize); (iv) which possesses resources of its own; (v) which is capable of perceiving its environment (but to a limited extent); (vi) which has only a partial representation of its environment (and perhaps none at all); (vii) which possesses skills and can offer services; (iix) which may be able to reproduce itself; (ix) whose behaviour tends towards satisfying its objectives, taking account of the resources and skills available to it and depending on its perception, its representation and the communications it receives.

In MAS, distributed planning is considered as a very complex task (Jung and Zelinsky,), (Sapena et al., 2008). In fact, distributed planning ensures how the agents should plan to work together, to decompose the problems into subproblems, to assign these subproblems, to exchange the solutions of subproblem, and to synthesize the whole solution which itself is a problem that the agents must solve (Ferrando and Onaindia, 2013), (Forget et al., 2008), (Ghallab et al., 2014). The actions of the other agents can induce a combinatorial explosion in the number of possibilities which the planner will have to consider, returning the space of research and the size of solution exponentially larger. There are several techniques to reduce data-processing complexity of planning interactions with other agents including (Au et al.,): (i) dividing states in the classes of equivalence, (ii) reducing search space into states which are really required. (iii) planning on line, i.e., eliminating the possibilities which do not emerge during the execution of plan.

This article is concerned with two important matters: how to define the MAS in a manner such that it has more utility to deploy it, and how to use such a MAS for the advanced software. The MAS must discover the action to be taken by supervising the application and its environment and analyzing the data obtained.

With MAS, we face two important matters: (i) the detection of a need for action. the need for action must be discovered by supervising the application and its environment and analyzing data obtained. (ii) the planning of the action. It consists to envisage the action (by proposing which modifications need to be made) and by programming it. In practice, the opposite dependency also requires consideration: Only those situations which can be repaired by an action taken which can really be planned should be considered during the analysis.

Gharbi A., Gharsellaoui H. and Ben Ahmed S..
Multi-Agent Control System.
DOI: 10.5220/0005001101170124
In *Proceedings of the 9th International Conference on Software Engineering and Applications* (ICSOFT-EA-2014), pages 117-124
ISBN: 978-989-758-036-9

This paper introduces a simple Benchmark Production System that will be used throughout this article to illustrate our contribution which is developped as agent-based application. We implement the Benchmark Production System in a free platform which is JADE (JavaTM Agent DEvelopment) Framework. JADE is a platform to develop multi-agent systems in compliance with the FIPA specifications (Vitabile et al., 2009), (Su and Wu, 2011), (Bordini and all., 2006).

In the next section, we present the Benchmark Production System. The third section introduces the formal representation of agent behavior. The fourth section presents the creation of JADE agents. We conclude in the last section.

2 BENCHMARK PRODUCTION SYSTEM

As much as possible, we will illustrate our contribution with a simple current example called *RARM* (Hrz and Zhou, 2007). We begin with the description of it informally, but it will serve as an example for various formalism presented in this article. The benchmark production system *RARM* represented in the figure 1 is composed of two input and one output conveyors, a servicing robot and a processing-assembling center. Workpieces to be treated come irregularly one by one. The workpieces of type *A* are delivered via conveyor *C*1 and workpieces of the type *B* via the conveyor *C*2. Only one workpiece can be on the input conveyor. A robot *R* transfers workpieces one after another to the processing center. The next workpiece can be put on the input conveyor when it has been emptied by the robot. The technology of production requires that first one *A*-workpiece is inserted into the center *M* and treated, then a *B*-workpiece is added in the center, and last the two workpieces are assembled. Afterwards, the assembled product is taken by the robot and put above the *C*3 conveyer of output. the assembled product can be transferred on *C*3 only when the output conveyor is empty and ready to receive the next one produced.

Traditionally, the *RARM* systems are directly controlled by a central server. The server proposes the schedule for the system as a whole and dispatches commands to the robots. This results is reliable and predicable solutions. The central point of control also allows an easier diagnosis of the errors. However, a variation in user's needs leads to change the centralized architecture. Customers ask more and more for self-management system, i.e., systems that can adapt their behavior with changing circumstances in

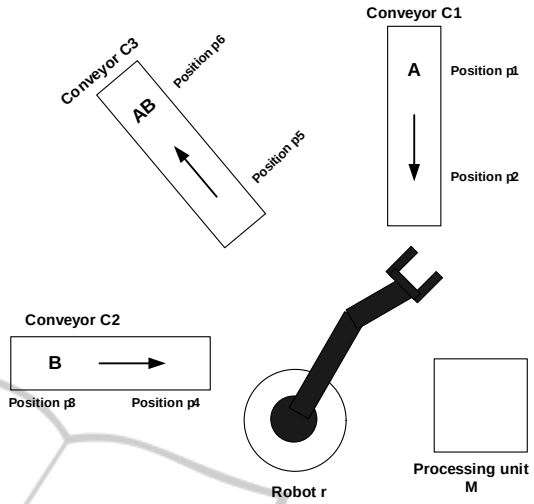

Figure 1: The benchmark production system RARM.

an autonomous way. Self-management with regard to the dynamics of system needs two specific quality requirements : flexibility and openess. Flexibility refers to the capacity of the system to treat dynamic operating conditions. The openess refers to the capacity of the system to treat robots leaving and entering system. To treat these new quality requirements, a radically new architecture was conceived based on multi-agent systems (Figure 2). Applying a situated multi-agent system opens perspective to improve the flexibility and the openess from the system: the robots can adapt to the current situation in their vicinity, order assignment is dynamic, the system can therefore treat in an autonomous way the robots leaving and reentring the system, etc. However, a decentralized architecture can lead to a certain number of implications, in particular distributed planning can have an impact on the total efficiency of the system. In fact, this critical topic must be considered during the design and development of multi-agent system.

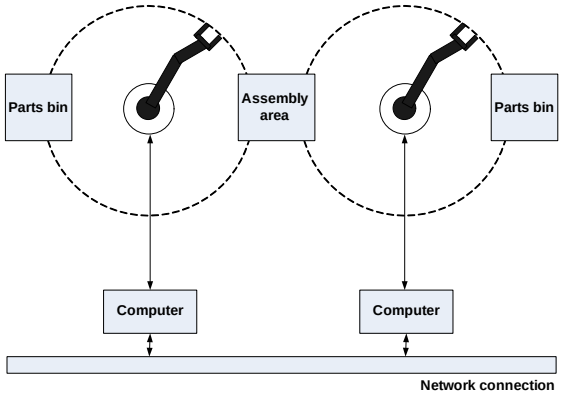

Figure 2: The distributed Production system.

3 THE FORMAL REPRESENTATION OF AGENT

We can define a Multi-Agent System (Ferber,) as (i) An environment E, that is, a space which generally has volume; (ii) A set of objects, O. These objects are situated, that is to say, it is possible at a given moment to associate any object with a position in E; (iii) An assembly of agents, A, which are specific objects (a subset of O), represent the active entities in the system; (iv) An assembly of relations, R, which link objects (and therefore, agents) to one another; (v) An assembly of operations, Op, making it possible for the agents of A to perceive, produce, transform, and manipulate objects in O; (vi) Operators with the task of representing the application of these operations and the reaction of the world to this attempt at modification, which we shall call the laws of the universe.

Running Example
For instance, if we consider RARM:
E = the physical space
A = the robots
O = other robots + objects (pieces,...)

3.1 Agent Conception

Perception is responsible for collecting runtime information from the virtual environment. The perception component supports selective perception, enabling an agent to direct its perception to its current tasks. The perception component interprets the representation resulting in a percept. A percept consists of data elements that can be used to update the agent's current knowledge (Figure 3).

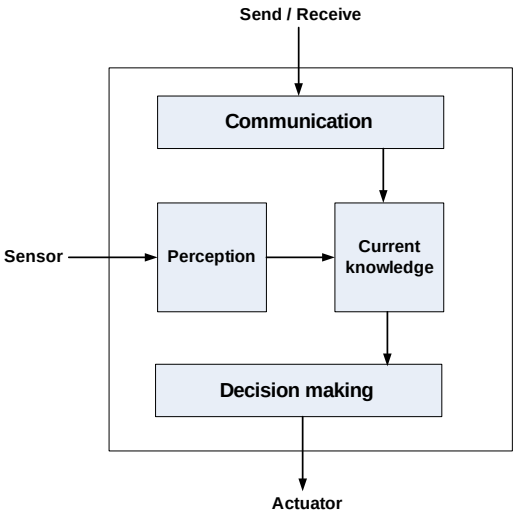

Figure 3: Interaction Agent.

The Decision making component encapsulates a behavior-based action selection mechanism. Decision making is responsible for realizing the agent's tasks by invoking actions in the virtual environment. To enable situated agents to set up collaborations, behavior-based action selection mechanisms are extended with the notions of role and situated commitment.

Communication is responsible for communicative interactions with other agents. Message exchange enables agents to share information directly and set up collaborations. The communication module processes incoming messages and produces outgoing messages according to well-defined communication protocols.

3.2 Formal Representation

Figure 4 shows a conceptual model of AI planning. The three components include (i) the planner, (ii) the plan-execution agent, and (iii) the world in which the plans are to be executed (the formal representation is based on the work (Ghallab et al., 2004).

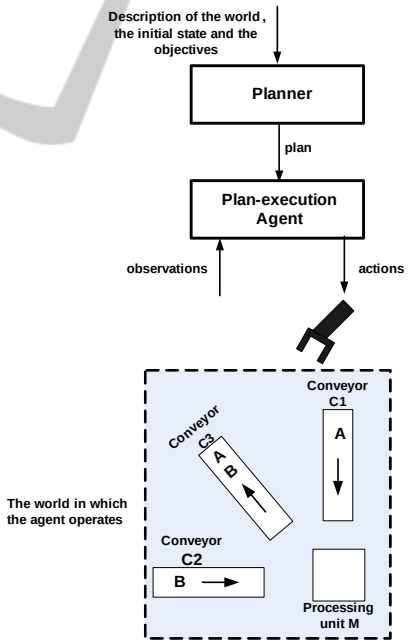

Figure 4: The Conceptual Model Planning.

The planner's input includes descriptions of Σ, the initial state(s) that Σ might be in before the plan-execution agent performs any actions, and the desired objectives (e.g., to reach a set of states that satisfies a given goal condition, or to perform a specified task, or a set of states that the world should be kept in or kept out of, or a partially ordered set of states that we might want the world to go through). If the planning

is being done online (i.e., if planning and plan execution are going on at the same time), the planner's input will also include feedback about the current execution status of the plan or policy. The planner's output consists of either a plan (a linear sequence of actions for the agent to perform) or a policy (a set of state-action pairs with at most one action for each state).

A classical planning problem is one that satisfies a very restrictive set of assumptions:

State-transition Model. The world is a finite state-transition system, i.e., a triple $\Sigma = (S; A; \gamma)$, where S is a finite set of states, A is a finite set of actions, $\gamma : S \times A \longmapsto 2^S$ is a state-transition function. If $\gamma(s; a) \neq \oslash$; then we say that a is applicable to s or executable in s.

Full Observability. Σ's current state is always completely knowable.

Determinism. For every s and a, $|\gamma(s;a) \leq 1|$. In other words, if a is applicable to s, then there is exactly one possible outcome, namely the state in $\gamma(s;a)$. Furthermore, there is exactly one initial state s_0 that will be Σ's current state before plan-execution begins.

Single Agency. The plan-execution agent is the only agent capable of making any changes in the world. If it were not for this agent's actions, the world would be static.

Achievement Goals and Sequential Plans. The planner's objective is to produce a plan (i.e., a linearly ordered finite sequence of actions) that puts Σ into any one of some finite set of states S_g.

Implicit Time: Actions have no duration; they are instantaneous state tran- sitions.

Offline Planning. The planner produces a complete plan for the given initial and goal states prior to any execution of its plan by the plan-execution agent. In multi-agent systems, Assumption 4 does not hold, and several of the other assumptions may not necessarily hold.

Running Example

According to figure 5 :

- *A set of positions* $\{p_1, p_2, \ldots\}$ *: A position is used to localise the workpiece A, B or AB;*
- *A set of robots* $\{r_1, r_2, \ldots\}$ *: Each robot transfers a workpiece one after one to be processed;*
- *A set of workpieces of type A* $\{a_1, a_2, \ldots\}$;
- *A set of workpieces of type B* $\{b_1, b_2, \ldots\}$;
- *A set of workpieces of type AB* $\{ab_1, ab_2, \ldots\}$;

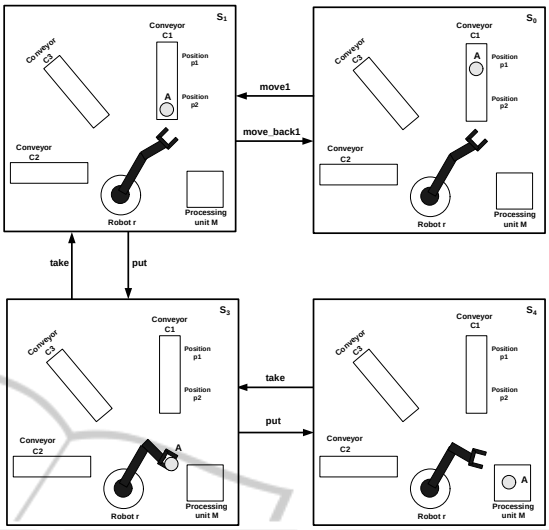

Figure 5: A state-transition for RARM.

- *A set of conveyors* $\{C_{1i}, C_{2i}, C_{3i}\}$; *: A conveyor* C_{1i} *(resp.* C_{2i}, C_{3i}*) is responsible for transfering set of workpieces of type A (resp B, AB);*
- *A set of processing Centers M* $\{M_1, M_2, \ldots\}$ *: first one A-workpiece is inserted into M and processed, then one B-workpiece is added into the center M, and last both workpieces are assembled.*

The set of states is $\{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}, s_{11}\}$ *where :*

- s_0 *(resp.* s_4, s_{10}*) is meaning a workpiece of type A (resp. B, AB) is at position* $p1$ *(resp.* $p3$, $p5$*);*
- s_1 *(resp.* s_5, s_{11}*) is meaning a workpiece of type A (resp. B, AB) is at position* $p2$ *(resp.* $p4$, $p6$*);*
- s_2 *(resp.* s_6, s_9*) is meaning a workpiece of type A (resp. B, AB) is taken by the robot r;*
- s_3 *(resp.* s_7, s_8*) is meaning a workpiece of type A (resp. B, AB) is put in the processing unit M.*

There are nine possible actions in the domain.

- *a workpiece of type A is trasnported from position p1 to position p2;*
- *the robot transports a workpiece of type A;*
- *the piece is put in the processing unit M;*
- *a workpiece of type B is trasnported from position p3 to position p4;*
- *the robot transports a workpiece of type B;*
- *the piece is put in the processing unit M;*
- *the robot picks up the assembled piece;*
- *the assembled piece is put on the conveyor C3;*
- *a workpiece of type AB is trasnported from position p5 to position p6.*

The set of actions is $\{move_1, move_2, move_3, move_4, move_5, move_6, take_1, take_2, take_3, load_1, load_2, load_3, put_1, put_2, put_3, process_1, process_2, process_3\}$ *where :*

- $move_1$ (*resp.* $move_2$) *is meaning a workpiece of type A is transported from position* p_1 (*resp.* p_2) *to position* p_2 (*resp.* p_1);
- $move_3$ (*resp.* $move_4$) *is meaning a workpiece of type B is transported from position* p_3 (*resp.* p_4) *to position* p_4 (*resp.* p_3);
- $move_5$ (*resp.* $move_6$) *is meaning a workpiece of type AB is transported from position* p_5 (*resp.* p_6) *to position* p_6 (*resp.* p_5);
- $take_1$ (*resp.* $take_2$, $take_3$) *is meaning the operation of taking a workpiece of type A (resp. B , AB);*
- $load_1$ (*resp.* $load_2$,$load_3$) *is meaning the fact of loading a workpiece of type A (resp. B , AB);*
- put_1 (*resp.* put_2, put_3) *is meaning the operation of putting a workpiece of type A (resp. B , AB);*
- $process_1$ (*resp.* $process_2$, $process_3$) *is meaning the fact of processing a workpiece of type A (resp. B , AB).*

The current configuration of the domain is denoted using instances of the following predicates, which represent relationships that change over time.

- *occupied*(c_1) *(resp. occupied*(c_2), *occupied*(c_3)): *conveyer c1 (resp. c2, c3) is already occupied by a workpiece of type A (resp. B, AB);*
- *empty(c1) (resp. empty(c2), empty(c3)): conveyer c1 (resp. c2, c3) is already ready to transport a workpiece of type A (resp. B, AB);*
- *at(r,p2) (resp. at(r,p4), at(r,p5)): Robot r is currently at position p2 (resp. p4, p5);*
- *loaded(r, a) (resp. loaded(r, b), loaded(r, ab)) : Robot r is currently loading the workpiece a (resp. b, ab) of type A (resp. B, AB);*
- *put(r, a) (resp. put(r, b),put(r, ab)): Robot r is currently putting the workpiece a (resp. b, ab) of type A (resp. B, AB);*
- *empty(r): the robot r is empty;*
- *empty(a) (resp. empty(b), empty(ab)): there is no workpiece of type A (resp. B, AB).*

NB Events are not chosen or triggered. If e is an event and $\gamma(s,e)$ is not empty, then e can possibly occur when the system is in state s.

Given a state transition system Σ, the purpose of planning is to find which actions to apply to which states in order to achieve some objectives when starting from a given situation. A plan is a structure that gives the appropriate actions. The objective can be specified by a goal state s_g or a set of goal states S_g. The objective can be obtained by any sequence of state transitions that ends at one of the goal states.

Running Example
If we want to have workpieces in the processing unit M, then the set of goal states is $S_g = \{s_3, s_7, s_8\}$

Plan: is any sequence of actions $\pi = (a_1, \ldots, a_k)$, where $k \geq 0$. The length of the plan is $|\pi| = k$, the number of actions. If $\pi 1 = (a_1, \ldots, a_k)$ and $\pi 2 = (a'_1, \ldots, a'_j)$ are plans, then their concatenation is the plan $\pi 1.\pi 2 = (a_1, \ldots, a_k, a'_1, \ldots, a'_j)$. The state produced by applying π to a state s is the state that is produced by applying the actions of π in the order given.
The plan π is executable in a state s_0 if there is a sequence of states $(s_0; s_1; \ldots; s_n)$ such that for i = 1; ...; n, $s_i = \gamma(s_{i-1}; a_2)$. In this case we say that $(s_0; s_1; \ldots; s_n)$ is π's execution trace from s_0, and we define $\gamma(s_0; \pi) = s_n$. If s_n satisfies the goal g, then we say that π is a solution for the planning problem P = (O; s_0; g).

Running Example
if s_0 *and* $g = \{$*workpiece in the processing unit*$\}$. *Let:*

- $\pi 0$: *(*$move_1$, $take_1$, $load_1$, $process_1$*)*
- $\pi 1$: *(*$load_1$, put_1, $process_1$, $move_2$*)*
- $\pi 2$: *(*$move_1$, $take_1$, $load_1$, put_1, $process_1$, $move_2$*)*

Then $\pi 0$ *is not a solution because although it is applicable to* s_0, *the resulting state is not a goal state;* $\pi 1$ *is not a solution because it is not applicable to so;* $\pi 3$ *is the only solution.*

4 CREATING JADE AGENTS

JADE is a Java tool and therefore creating a JADE-based multi-agent system requires creating Java classes. For more details, we refer to (Bellifemine et al., 2010b), (Caire, 2009), (Bellifemine et al., 2010a), (Bellifemine et al., 2004). Creating a JADE agent is very easy through defining a class that extends the *jade.core.Agent* class and implementing the *setup*() method. Each class introduced in the Figure 6 will be presented in the following paragraphs.

Running Example
The *setup*() method is invoked when agent starts running and permits to initialize instance variables, register agent and attach one or more behaviors to the agent.

```
import jade.core.Agent;
public class Robot extends Agent {
protected void setup() {
System.out.println("Hello everybody! I am an
```

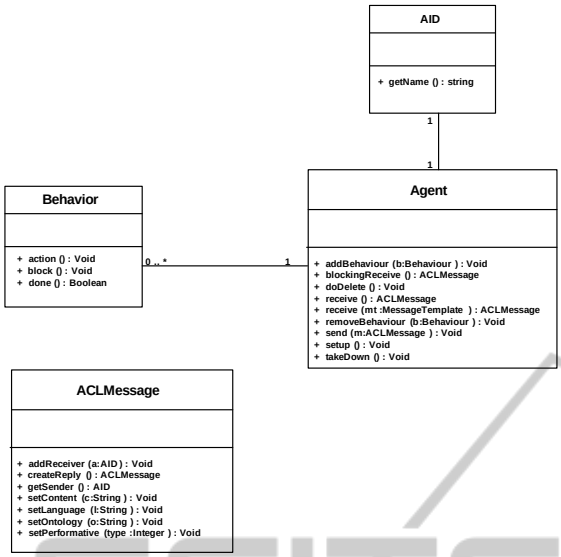

Figure 6: JADE agent.

agent");
}
}

4.1 Agent Identifier

Each agent is identified by an agent identifier represented as an instance of the jade.core.AID class. The getAID() method of the Agent class allows retrieving the agent identifier. An AID object includes a globally unique name plus a number of addresses. The name in JADE has the form $< nickname > @ < platform - name >$ so that an agent called *Robot*1 living on a platform called *RARM* will have *Robot*1*@RARM* as globally unique name. The addresses included in the AID are the addresses of the platform the agent lives in. These addresses are only used when an agent needs to communicate with another agent living on a different platform.

4.2 Agent Discovery

The *JADE* platfrom allows the possibility to discover dynamically the available agents. To do so, a *yellowpages* service permits agents to describe one or more services they provide. An agent can register (publish) services and search to discover services.

Running Example
In order to publish a service, an agent must create a proper description which is an instance of *DFAgentDescription* class and call the *register*() method of *DFService* class.
/// Register the Robot in DFService DFAgentDescription dfd = new DFAgentDescription();
dfd.setName(getAID());
ServiceDescription sd = new ServiceDescription();
sd.setType("Robot");
sd.setName("Robot-executing");
dfd.addServices(sd);
try {
DFService.register(this, dfd);
}
catch (FIPAException fe) {
fe.printStackTrace();
}

It is possible to search some agents, if the agent provides the DF with a template description. The result of the research is a list of all the descriptions matching the template.

Running Example
The *search*() method of the *DFService* class ensures the result.
DFAgentDescription template = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType("Robot");
template.addServices(sd);
DFAgentDescription[] result ;
try {
do
{
result = DFService.search(myAgent, template);
robotAgents = new AID[result.length];
for (int i = 0; i < result.length; i++) {
robotAgents[i] = result[i].getName();
}
}
while (result.length <= 0);
}
catch (FIPAException fe) {
fe.printStackTrace();
}
nbRobots=robotAgents.length;

4.3 Message Exchanged between JADE Agents

Agents never interact through method calls but by exchanging asynchronous messages. Obviously, inter-agent interaction will be very difficult until all agents adopt the same communication language, and fortunately ACL standards ensure this requirement. All JADE agents communicate using messages that obey the FIPA ACL specification, which is described in : http//www.fipa.org.
This format comprises a number of fields and in particular: (1) the sender of the message, (2) the list

of receivers, (3) the communicative intention (also called performative indicating what the sender intends to achieve by sending the message (for example the performative can be REQUEST, INFORM, QUERY_IF, CFP (call for proposal), PROPOSE, ACCEPT_PROPOSAL, REJECT_PROPOSAL, and so on). (4) The content i.e. the actual information included in the message which may be string in simple cases; otherwise we need a content language, a corresponding ontology, and a protocol. (5) The ontology i.e. the vocabulary of the symbols used in the content and their meaning (both the sender and the receiver must be able to encode expressions using the same symbols to be sure that the communication is effective)

4.3.1 Sending a Message

Sending a message to another agent is as simple as filling the fields of an *ACLMessage* object and then call the *send*() method of the Agent class. The code below informs an agent whose nickname is Robot1 that the production must be decreased.

Running Example

```
ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
msg.addReceiver(new AID("Robot1", AID.ISLOCALNAME));
msg.setOntology("Production");
msg.setContent("We must decrease in the production");
send(msg);
```

4.3.2 Receiving a Message

As mentioned above the JADE runtime automatically posts messages in the receiver's private message queue as soon as they arrive. An agent can pick up messages from its message queue by means of the *receive*() method. This method returns the first message in the message queue (removing it) or null if the message queue is empty and immediately returns.

Running Example

```
ACLMessage msg = receive();
if (msg != null) {
// Process the message
}
```

4.3.3 Blocking Behavior Waiting a Message

Some behaviors must be continuously running and at each execution of their *action*() method, must check if a message is recceived and perform some action.

Running Example

```
public void action() {
ACLMessage msg = myAgent.receive();
if (msg != null) {
// Message received. Process it
...
}
else {
block();
}
}
```

4.3.4 Selecting a Message

When a template is specified, the *receive*() method returns the first message (if any) matching it, while ignores all non-matching messages. Such templates are implemented as instances of the *jade.lang.acl.MessageTemplate* class that provides a number of factory methods to create templates in a very simple and flexible way.

Running Example

The *action*() method is modified so that the call to *myAgent.receive*() ignores all messages except those whose performative is REQUEST:

```
public void action() {
MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.REQUEST);
ACLMessage msg = myAgent.receive(mt);
if (msg != null) {
// REQUEST Message received. Process it
...
}
else {
block();
}
}
```

5 CONCLUSION

Distributed planning is narrowly interlaced with the distributed resolution of the problems, being a problem in itself and means to solve a problem. The main aim of this paper is how to ensure a distributed planning in Multi-Agent System (MAS) composed of several intelligent autonomous agents able to take the initiative instead of simply reacting in response to its environment. We create the MAS through JADE platfrom and show the interaction between the different agents through exchanging messages. All our contributions are applied on the benchmark production system (RARM system).

REFERENCES

Au, T.-C., Kuter, U., and Nau, D. Planning for interactions among autonomous agents.

Bellifemine, F., Caire, G., and Greenwood, D. (2004). Developing multi-agent systems with jade.

Bellifemine, F., Caire, G., Trucco, T., and Rimassa, G. (2010a). Jade programmers guide.

Bellifemine, F., Caire, G., Trucco, T., Rimassa, G., and Mungenast, R. (2010b). Multi-agent-system-based decentralized coordinated control for large power systems. *Jade Administrators Guide*.

Bordini, R. and all. (2006). A survey of programming languages and platforms for multi-agent systems. *Informatica*, 30(1):3344.

Caire, G. (2009). Jade tutorial : Jade programming for beginners.

Ferber, J. Multi-agent system: An introduction to distributed artificial intelligence. *Harlow: Addison Wesley Longman*.

Ferrando, S. P. and Onaindia, E. (2013). Context-aware multi-agent planning in intelligent environments. *Information Sciences*, 227(1):22–42.

Forget, P., DAmours, S., and Frayret, J.-M. (2008). Multi-behavior agent model for planning in supply chains: An application to the lumber industry. *Robotics and Computer-Integrated Manufacturing*, 24(5):664–679.

Ghallab, M., Nau, D., and Traverso, P. (2004). Automated planning. *International Journal of Electrical Power & Energy Systems*.

Ghallab, M., Nau, D., and Traverso, P. (2014). The actor's view of automated planning and acting: A position paper. *Artificial Intelligence*, 208(1):1–17.

Hrz, B. and Zhou, M. (2007). Modeling and control of discrete-event dynamic systems with petri nets and other tools. page 67.

Jung, D. and Zelinsky, A. An architecture for distributed cooperative planning in a behaviour-based multi-robot system. *Robotics and Autonomous Systems*, 26(2).

Liu, B., Su, H., Li, R., Sun, D., and Hu, W. (2014). Switching controllability of discrete-time multi-agent systems with multiple leaders and time-delays. *Applied Mathematics and Computation*, 228(1):571–588.

Sapena, O., Onaindia, E., Garrido, A., and Arangu, M. (2008). A distributed csp approach for collaborative planning systems. *Engineering Applications of Artificial Intelligence*, 21(5):698–709.

Su, C.-J. and Wu, C.-Y. (2011). Jade implemented mobile multi-agent based, distributed information platform for pervasive health care monitoring. *Applied Soft Computing*, 11(1):315–325.

Vitabile, S., Conti, V., Militello, C., and Sorbello, F. (2009). An extended jade-s based framework for developing secure multi-agent systems. *Computer Standards & Interfaces*, 31(5):913–930.

xia Dou, C., wei Hao, D., Jin, B., qian Wang, W., and An, N. (2014). Multi-agent-system-based decentralized coordinated control for large power systems. *International Journal of Electrical Power & Energy Systems*, 58:130–139.