FROM ENGINEERING TO PROGRAMMING: SMART MULTI AGENT SYSTEM APPLICATIONS USING ARL

FROM ENGINEERING TO PROGRAMMING: SMART MULTI AGENT SYSTEM APPLICATIONS USING ARL

Salaheddin J. Juneidi
Computer Engineering Department
Palestine Technical University Khadoori
Hebron –West Bank
Palestine
Salaheddin.juneidi@ptuk.edu.ps

ABSTRACT: Modern computing systems are generally based on embedded smart agents. They are introduced as a new trend of computing paradigm; in which we can define smart entities in software system or robotic machines. This modern trend of technology calls for new approaches in both software engineering and programming techniques. Unlike Object Oriented programing languages, Agent oriented programming languages and agent oriented engineering are not stable and are not well defined, on the contrary, object orientation is well defined and consistent. From the fact that, agent oriented programming is an implementation of agent oriented engineering; this article follows this sequence and it tackles the application view over agent oriented software system. (Agent Role Locking) ARL theory is used to design and implement agents in software system, on the other hand *Java threads* are used to implement agents. The main aim is to show innovative incorporation relationship between engineering methodologies and programming application in Smart Agent Orientated Technology.

Keywords: *Agent Role Locking Theory (ARL), Java Threads, Agent class, Role Class, Agent-Oriented Programming AOP*

1. INTRODUCTION

1.1 Objects and agents: Bottom lines

Several comparisons took place between agents and objects as software entities, with the aim to have a clear view toward objects and agents. Actually, most of these differentiations give no absolute bottom lines between objects and agents regarding definition and application. It is worthy to have these bottom lines between objects and agents, to have a final decision on when and where agents can be embedded in our software; we mostly concern about is the minimum requirements needed for this intelligent software entity to be considered as an agent. In [1, 2] has discussed three requirements:

i. ***Autonomy***. An agent is not passively subject to a global, external flow of control in its actions. That is, an agent has its own internal thread of execution, typically oriented to the achievement of a specific task, and it decides for itself what actions it should perform and at what time.

ii. ***Situatedness/ adaptability***. Agents perform their actions while situated in a particular environment. The environment may be a computational one (e.g., a

Website) or a physical one (e.g., a manufacturing pipeline), and an agent can sense and effect some portions it.

iii. ***Proactivity*.** Agents accomplish their designed objectives in a dynamic and unpredictable environment the agent may need to act to ensure that its set of goals are achieved and that new goals are opportunistically pursued whenever appropriate

1.2 Agent Oriented Programming Languages

Various programming languages have been proposed to represent agent software entities. As a matter of fact, those languages and programming codes directed towards single agent base particular problem solution (*single problem solution).* This article is about to discuss a general agent base computing theory that has contemporary and complementary software solutions that take in consideration engineering and programming parts agent oriented software application, in other words, Agent Base Modeling Software (ABMS) as clarified in Figure 1. In past two decays new kinds of fifth generation languages [3, 4], agent-base libraries and agent tools have appeared to represent agent-based computing, on the other hand, there are a growing number of agent-based applications, those applications distributed in a variety industrial fields and software informatics. Agent-based modeling can be used to study dynamic and unpredictable phenomena, those phenomena such as self-control arranging work of machines; other example is to predict consumer choice for merchandise. Many other examples can be given like to have real applications that need to decide and act autonomously such as social network analysis, behavioral economics in stock market analyses, understand how people's behavior affects activities like pedestrian movement, transportation traffic as part of machine learning [5,6] …etc.

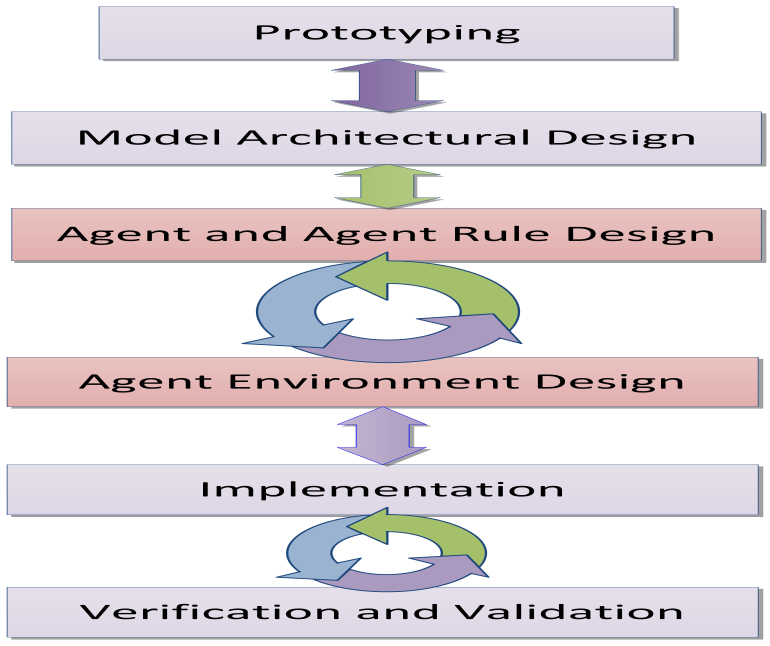

Prototyping
Model Architectural Design
Agent and Agent Rule Design
Agent Environment Design
Implementation
Verification and Validation

Figure 1 : Agent Base Modeling Software (ABMS

2. AGENT ROLE LOCKING THEORY ARL[1]

ARL [10] introduces a new view of multi-agent systems structure. The main concept is based on role decomposition and an agent entity plays only one role at a given time . The first step of analyzing MAS is to define the boundaries of a system, by these boundaries we define the *environment* we working with, the MAS environment is set by all functions and requirements specification of a given system. The next step is to define the *organizations* belong to this environment, the organizational view is recognized by defining the most general objectives of the system, within each organization, there are associated objectives and tasks, these objectives and tasks are performed by positions with responsibilities, Figure 2 for MAS analysis, which views of MAS environment into 1..a organizations, each organization is decomposed into various number of *super role* which represents a position with an authority (*permissions)* which represented in Figure 2 as a key, only agents which have certain keys can locked into legitimate certain roles in the organization, role's decomposition process proceeds until we reach *atomic roles* which represents the specific MAS functionality (*objectives)*, the atomic role performers are agents

entities (*responsibilities*) .this ARL view of MAS, as organizational view and role decompositions, is consistent with role definition in [7]:

i. **Objectives**: is represented in ARL by atomic role entity.

ii. **Responsibilities:** ARL specifies agent class entity as responsible to perform roles.

iii. **Permissions:** eligibility constrains within agent / role classes to be activated (locked), represented in ARL by role/ agent keys.

ARL has other important distinctions from other methodologies proposed for AOSE, by defining *agent types*, each agent type is identified by agent class (see next on agent stat[1]ic structure) that can be instantiated. Agent in MAS are not equal, they are different according to their goals, permissions entitle the agent to have more (*keys)*.

ARL presents two models (*static model and dynamic model)* to describe the structure and the behavior of agents and the roles they locked into, the next two sections have summary of those two models.

2.1 Super & Atomic roles

ARL emphasises on an individual agent can perform a single role in a given time. To manage agent / role complexity and mobility, it's important to determine the granularity of each role. Because as shown in figure 3 agents are can lock /unlock (to/from) atomic roles according to constrains discussed later in ARL theory.

Actually, during system design and implementation, we cannot count on a generic role to capture agent/ role as an engineering process, for instance, if an agent is specified to play the role of *studying at university*, (figure 4) agents are autonomous and can change roles any time or according there internal state, and *studying in the university* has various activities (dependent and independent) that have communication with different agent [8]. On that fact roles must be fine grained an agent may play (locked with) this called atomic role that normally has defined specific activities

[1] Going into details of ARL analysis and design of MAS is beyond this paper, consider references [7,9] for more about ARL

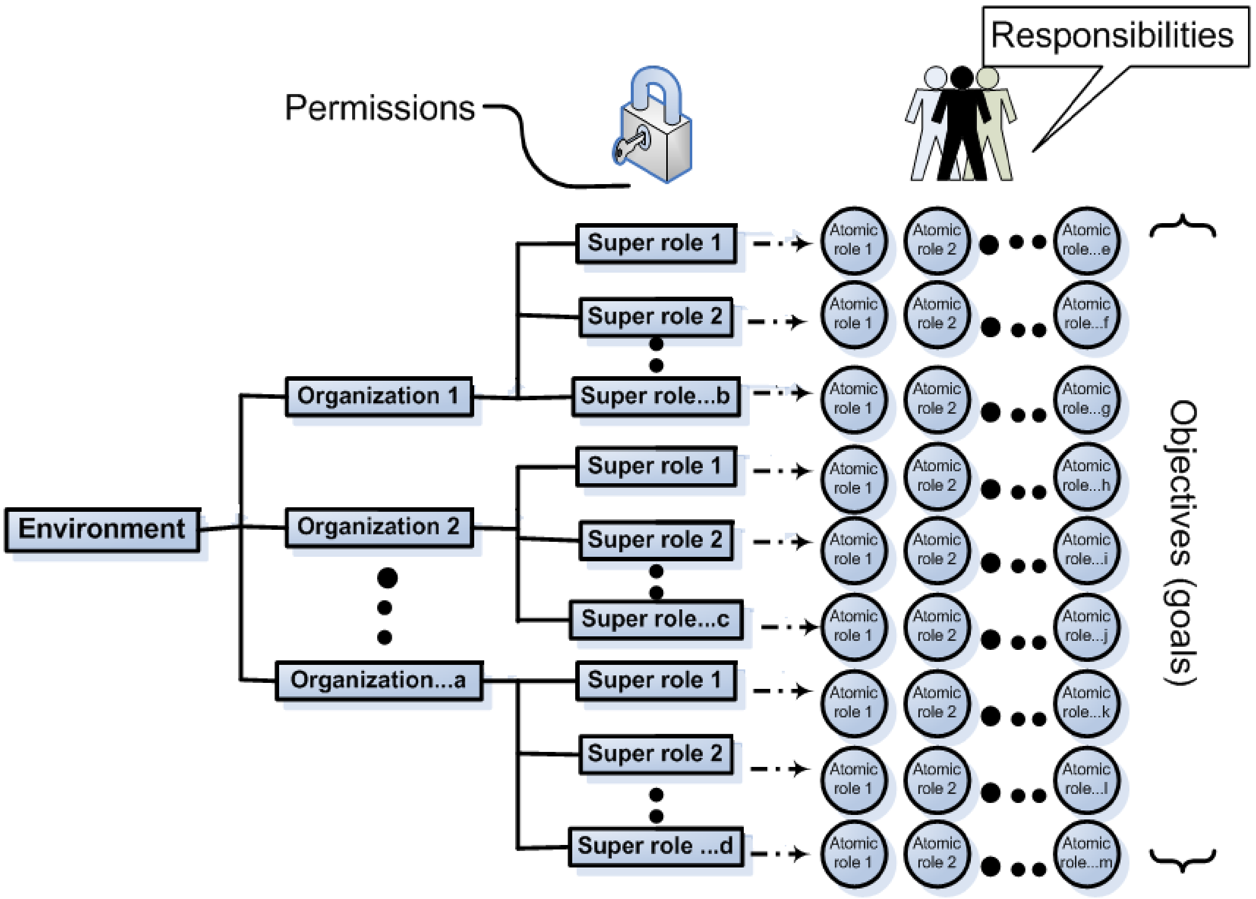


Figure 2: An overview of ARL on MAS environment decompositions and role definition

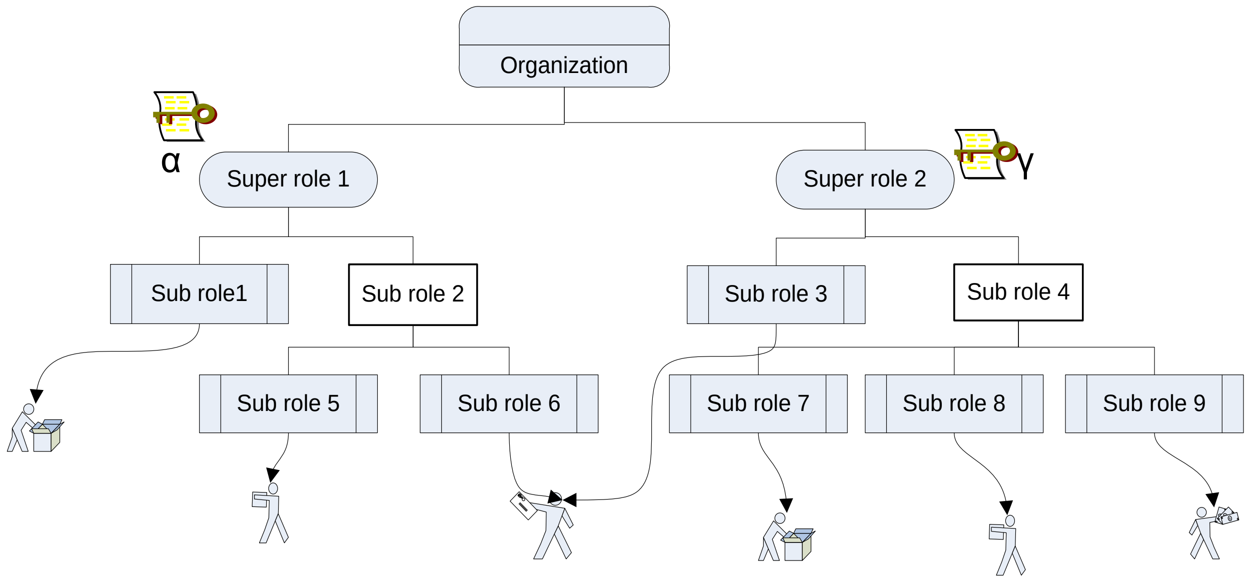


Figure 3: general organizational view of MAS with agents and keys

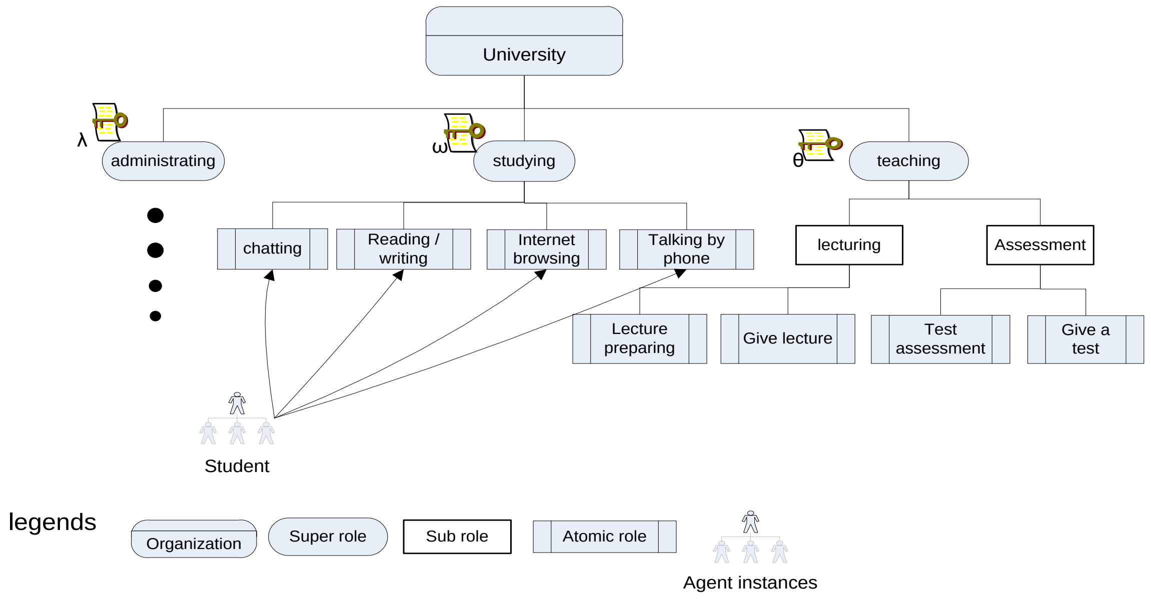


Figure 4 : University Organization breaking up into *Super and Atomic Roles*

2.2 Agent definition

Each agent is given authorization and eligibility to play atomic roles, this eligibility is presented by super role assigned *key*, so *agents* can lock into any *atomic role* belongs to that super role.

Organizational structure in MAS must be defined, in which it clarifies super and atomic roles. Then we can define Agent types. For each super role an agent type can be generated. So for super roles in Figure 3 we have three agent types *Administrator*, *student*, and *teacher*, each of which have the right and permissions to perform the corresponding atomic roles, so these agent types have keys (λ, ω, θ consecutively) of super roles to perform their atomic roles. These agent types may have other permission from other organization in MAS environment, accordingly, as much as keys added to super roles, as much permission are granted. For an agent type as much as keys it has, as much as atomic roles it can locked though MAS environment, all that locking and unlocking take palce according to MAS functionality specifications.

Returning back to our student agent example, the role decomposition in Figure 3, gives us manageable way to understand what the role (super role) and activities that student instance may perform in a given time. In our example the student agent must have (ω) key to able to lock into its appropriate atomic role.

ARL supports dynamic agent environment, there are no mapping or pr-planned scenarios among roles and agents, Agents lock into atomic roles by two methods:

i. ***Role launching**:* agent launches atomic role according its internal state, functional requirement, or time bases.

ii. ***Role satisfying:*** agent performs atomic role interdependent with an atomic role that been launched by another agent.

Table 1 gives an example of interdependent roles for the university organization, first column is launching atomic roles that need another roles to interact with that defined in second column. This table called *Agent Role Coupling* (*ARC*) that defines the interdependency between roles in system.

Because an agent may have keys to lock into several atomic roles in different organizations, because agents move through roles according internal state and functionality , so for a given unit of time ***t*** we cannot guarantee a specific agent would be existed in an organization . In the next subsections we are going through details of ARL definitions and assumptions.

Atomic role (launching)	**Key**	**Atomic role (satisfying)**	**key**
Studding (Reading , Chatting , Internet Browsing… etc	ω	Teaching (Lecture preparing , Give lecture, Give Test, Asses ..etc)	Θ
Teaching (Lecture preparing , Give lecture, Give Test, Asses ..etc)	θ	Studding (Reading , Chatting , Internet Browsing… etc	Ω
Administration (admission, assign classes, arrange classes …etc)	λ	Studding (Reading , Chatting , Internet Browsing… etc M	Ω
Administration (admission, assign classes, …etc)	λ	Teaching (Lecture preparing , Give lecture, Give Test, Asses ..etc)	Θ

Table 1: ARC table, some selected interdependent atomic roles

2.3 Static Model

ARL proposes new development on static structure of *Agent-Clas*s and *Role-Class* as two separate idle entities, none of these two classes are active, but when getting together (locked) , they become active, by this representation we can capture static systems with mobile intelligent entities. This kind of new static view development could be integrated with UML reaching to AUML [7,9] that supports object class on Agent Orientation paradigm we have *agent class* and *role class*. By this way agent entity is free from any role burden, it can move from one role to another without any pre-assigned agent-role mapping, agents entities can be instantiated to perform atomic roles, agents can move freely and instantiated according system functionality constrains (agent –role switching constrains – see ARL assumptions next section) or according to an agent internal / mentality state. **Role-Class:** represents the responsibilities of position, in which conclude the objectives that satisfied by an agent perform (*locked into*) according given permissions. **Agent -Class**: this class is mostly concern with agent side and its characteristics, like mental and internal state, goals, what kind of roles it can perform according to the knowledge the agent entity owns

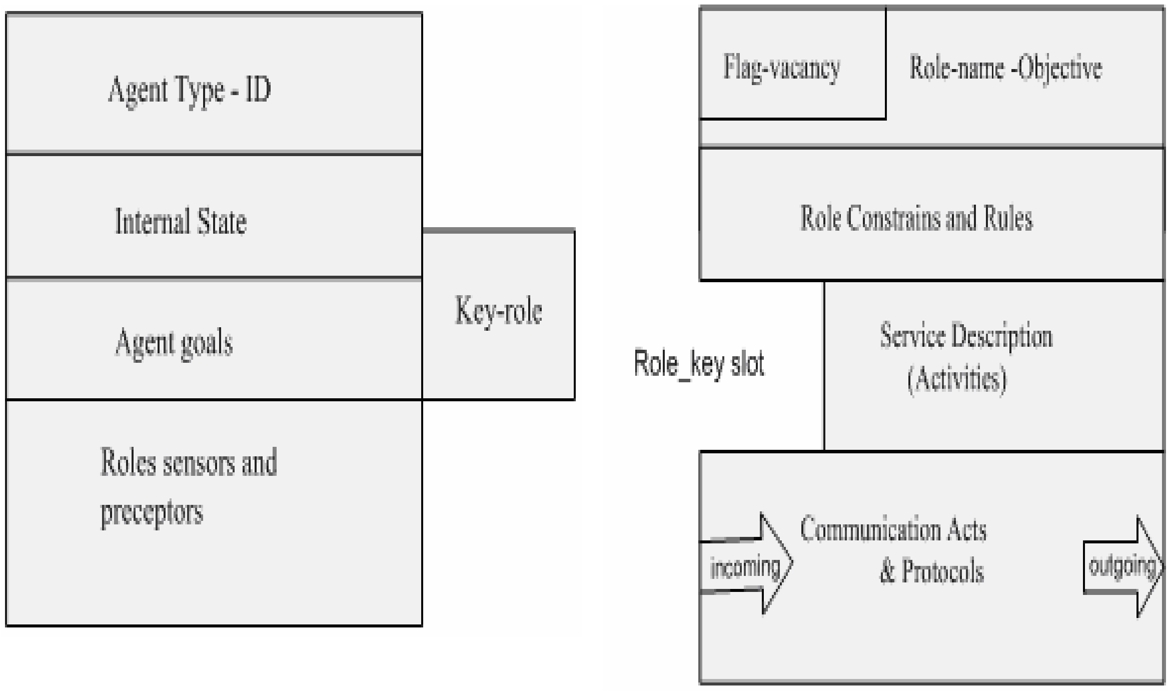

Agent Type - ID
Internal State
Agent goals
Roles sensors and preceptors
Key-role
Flag-vacancy
Role-name -Objective
Role Constrains and Rules
Role_key slot
Service Description (Activities)
Communication Acts & Protocols
incoming
outgoing

Figure 5 : Agent class (left) Role class right see [9]

2.4 Dynamic Model

Which involves in collecting interdependent atomic roles presented into *Atomic Roles Couples* table *(ARC) as table 1* (see ARL references [7,9]), this table defines the couples of interdependent roles and their candidates performing agents. *Interdependent role* contains dependent atomic role in one super role that couldn't be accomplish without the other role interaction *(communication*) . In Figure 4, say X and Y are interdependent super roles, for X the agent who eligible for launching this super role must own **α** key, for the Y super role the satisfying agent must own **β** key . To specify the details of interaction between roles couples, the Agent Interaction Protocols (AIP) is used.

AIP firstly presented [7,9]. Roles couples are represented by pairs; which represent MAS interaction, communication and functionality. From the agent point of view, the system start functioning when some agent instance ***lock into*** and launch a given role, which communicates an interdependent role that calls a satisfying agent, the two agent roles couples start to operate, other agents instances start activated and deactivated within roles according functional constrains, agent mental state, and agent-role switching constrains. **Role -Class**:

the role class supports *super roles*, it contains atomic objectives, rules and services description activities, as well as communication acts (CA) for another other atomic role see figure 6 of AIP for two interdependent roles X, Y and the agents to perform[9,10].

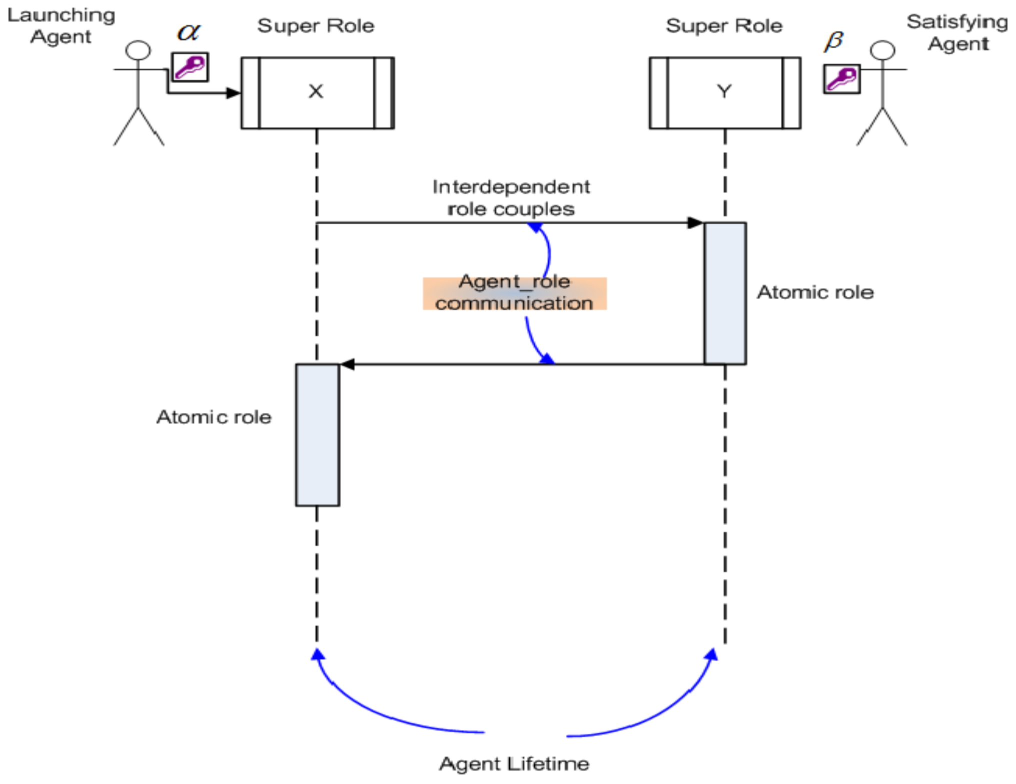


Figure 6 : Simple agent role AIP diagram

2.5 Definitions and Assumptions in ARL

i. An agent instant considered as agent type is independent from role entity ,agent entity is idle and activated when locked into role using given key.

ii. Running agent-role (and therefore, for the system to run) an agent must launch \ satisfy a role.

iii. Each agent class represents a unique agent type.

iv. An agent class can encapsulate knowledge describing agents' internal state, goals, intentions, preferences etc, as well as methods for role performing, and mechanisms for reasoning. The agent class' degree of **autonomy**, **flexibility** and **pro-activeness** depends on the *amount of knowledge and capabilities specified*, depending on the developers' decisions to meet the software functional and non-functional requirements.

v. An agent class can have as many as instances depending on functionality during system's run time. Agents instance inherits the same permissions and goals given to its agent's type.

vi. As agents, atomic ***roles*** are not active too. Atomic roles are represented by means of role classes. For a role to be performed (i.e. for a role class to become active) an agent has to lock into it.

vii. There is no direct interaction between the agents. For agents to interact, they need to be locked in roles with interdependent activities. Therefore, interaction, as well as any other activity, is carried out only when agents perform roles.

viii. When an agent locks to (launch) a given role, then the role with interdependent activities notify the MAS society (via its Vacancy Flag, as it will be specified in role class) that need to be served.

ix. For each atomic role to be served there must be at least one agent type attains the key of super role in which these atomic roles are belonging to.

x. Agents may create instances (internal instantiation) of its type to perform an atomic role. Typical situations that this may happen when other atomic roles need to be served simultaneously by the same agent. An agent instance –clone- terminates As soon as it accomplishes its atomic role.

xi. An agent instance performs one and only one atomic role at a given time.

xii. Agent - role switching constrains: An agent may unlock from one role to lock into another with respect to the following constrains :

a) Passive Sensing: Performing unlocking and locking according functional events and constant time interval.

b) Active Sensing (role satisfying): An agent, while performing a role, receives stimuli from another role, like vacancy flag , so it may decide to unlock from the current role to perform another role according to the stimuli.

c) Internal state and goal precedence (role launching): An agent may reach to the decision to unlock and lock into other role by reaching some internal state and goals that dictate it to do so. Alternatively, an agent goal may get a higher priority than the role objective it serves.

The value of this article is the application part, which is to apply ARL into real real application of MAS , starring from engineering designs of Agent Base System to writing programming and codes for that system, and show the increase of acceptance and the practical usability of AOSE to be considered as a new *computing paradigm*. As it provides overall view of MAS dynamic and static structures, presenting various modeling tools for agents and roles , as we going to see in the next sections , we will attest an application of agent systems using ARL on *multithreads programming language*.

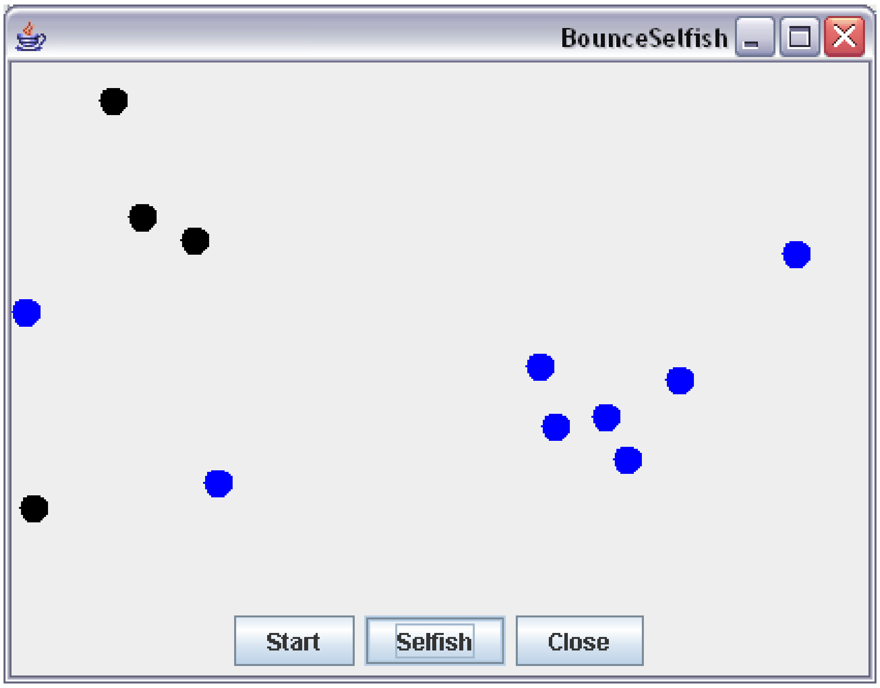


Figure 7: Bouncing ball agents moving randomly in frame

3. ARL APPLICATION

ARL provides a new-look for MAS, separating agent classes and the role classes -" *breaking complexity though decomposition*" - which make it possible to control agents separately and then join agent with roles that may eligible to perform. Java is object oriented programming language, in which similar to agent oriented if we add intelligent factors on it, on this assumption java as language will be used to program agents and roles classes. Most important we need to conserve agent's characteristics mentioned previously which are (*autonomous, proactive, mobile).*

We going to use simple example (*smart moving balls*) that will show diversity of agents behavior : which basically is throwing balls in some boundaries of canvas (form) , as soon as the balls (agents) created, they perform a role which is : moving freely in canvas boundaries with no collision with other balls , the knowledge that given to the agents is to check next available contagious position, some balls are selfish that is if the indented moving position is not available it force the agent (ball) allocated that position to move or else sleep (deactivate) for some time and so on , all balls can move as soon as there instant created so we can see more than thread agent(ball) runs in a one (*time slice*) unit of time depending on system functionality. The result as shown in Figure 7 describes free autonomous and non-stop moving balls and selfish balls (blue) within environment (Canvas) with no pre-planned next position because it depends on agent intelligence and contagious environments that has many other balls moving around. The most important idea is the agents and the environment are interacting and running according to their periodic circumstances without any forced or pre-planned schedule, and even without central or external interfere or control

on this example we have two types of agents (smart balls) one is normal and the second is selfish , and we have only one super role which is *Move* that has three atomic roles : *check_next_position ( ) , do_move(), and wait_ time().* Figure 8 depicts the ARL dynamic view through AIP by presenting moving one agent(ball) from one position to another. This AIP represents the general case of agent moves from one position to another , during execution.

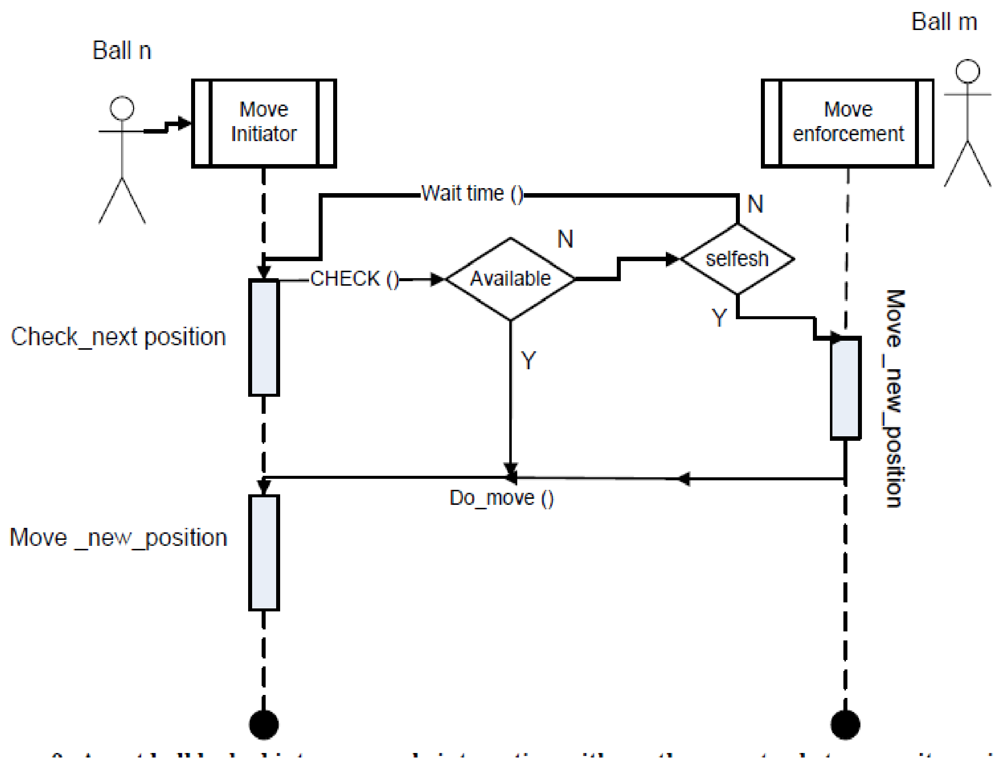


Figure 8: Agent ball locked into move role interacting with another agent role to commit moving

3.1 Agent and Threads in Java

ARL suggests to use ***Threads*** in Java programming language , as a thread has very much in common with agent as runnable entity, respectively, multi-threading techniques is used to represent MAS, the thread can grab the chance to execute the code in their *run* ( ) procedures.

(Figure 9.) . As an agent can be represented by thread, it can be activated, deactivated and terminated, the usual way to do this is through the static sleep ( ) and run () methods. In our example the run method of the BallThread class uses the call to sleep(5) to indicate that the thread will be idle for the next five milliseconds. After five milliseconds, it will start up again, but in the meantime, other threads (agent) have a chance to get work done.

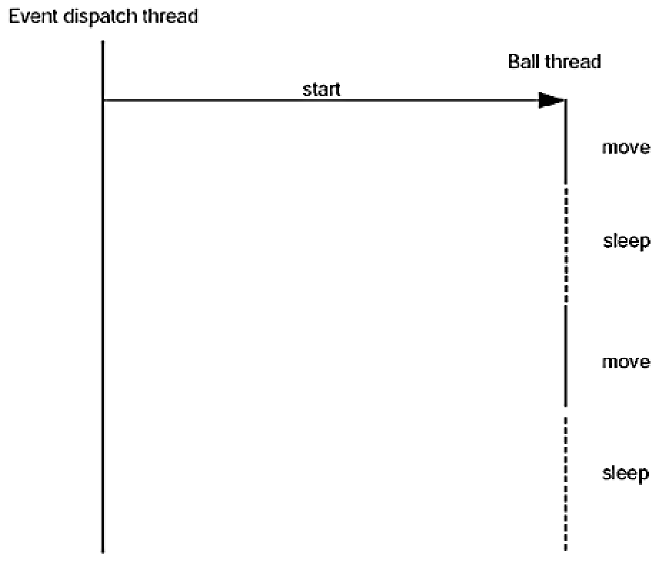


Figure 9. The Event Dispatch and Ball Threads in java

3.2 Thread Properties

3.2.1 New threads

Creating a thread with the new operator, practically means creating an agent instant for example, *new Ball()*— the thread is not yet running. This means unlike normal object oriented programming, in which execution starts as the main() or init () main program start running . But in agent orientation the execution will starts only when an agent is activated by locking into a legitimate role, that's the actual definition the new state of runtimeOF agent computing theory. So threads are allocated in RAM according but they will be run and activated only when locked to initiate a role or locked into interdependent running agent-role. Dependently it is not weird to run agent base system and it do nothing until new state and circumstances applied to start execution.

3.2.2 Runnable threads

Once the agent locked into role it invokes the run () method, and become the thread is runnable. A runnable thread may not yet be running according to agent (thread) behavior and role (routine /procedure) constraints . It is up to agent –role and MAS environment to give the thread time to run. When the code inside the thread begins executing, the thread is running. that a runnable thread may or may not be running at any given time. (This is why the state is called "runnable" and not "running.") See Figure 10 defines in Java threads' running options , the main issues in these options are priorities and blocking , this is how multi-threading is applied in Java program runtime.

3.3 Agent application using Java based on ARL

In our application , smart balls are bouncing in a given frame randomly , two kinds of balls provided : black and blue . no ball must collide with another at any giving time and no ball should go out of frame. The blue ball (selfish) has more priority to move freely. the start button will generate black ball (normal) the selfish button will generate blue ball (selfish). Balls can be generated as much as the system is exhausted, and no more RAM available.

The structure of balls is represented of JAVA threads and the positions of these balls are stored in array list objects as the following code:

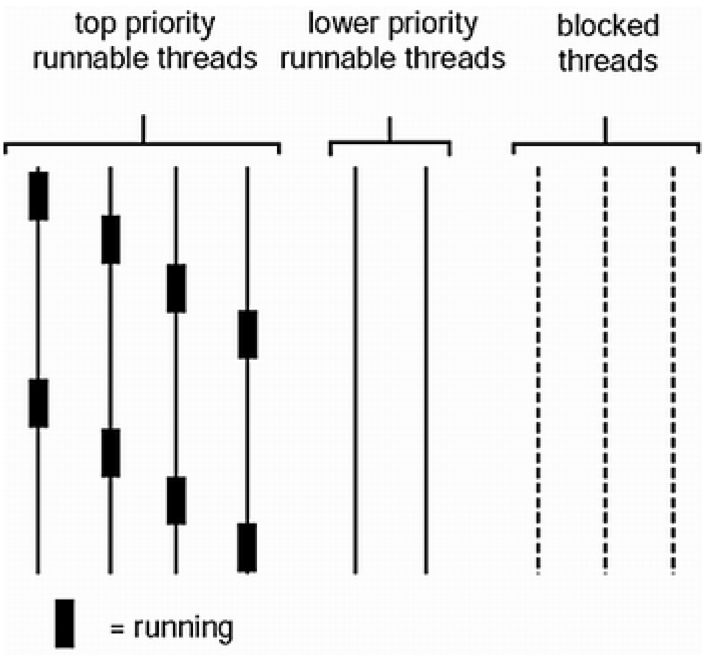
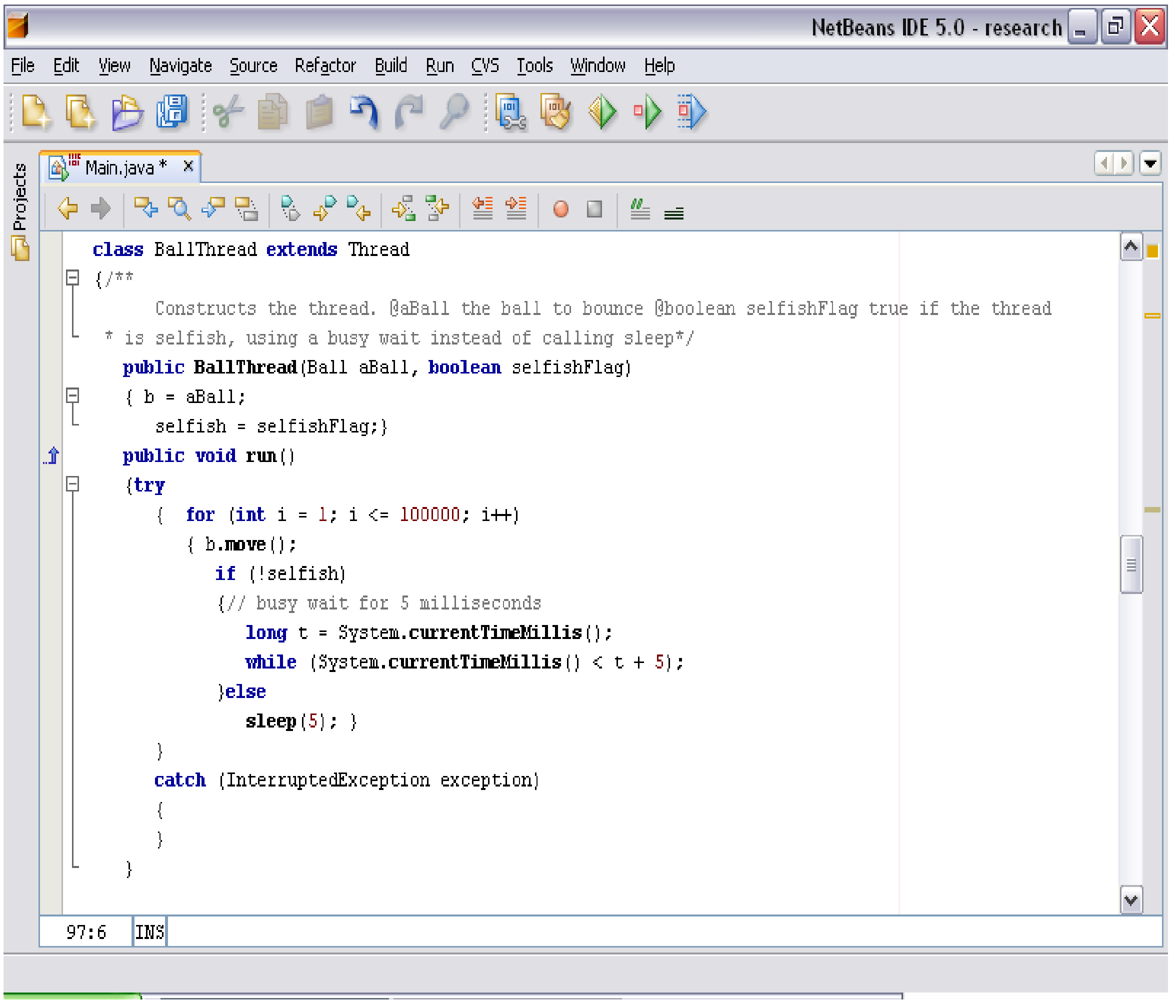

top priority runnable threads
lower priority runnable threads
blocked threads
= running

Figure 10. agents (Threads) in Java language runtime

```
public void addBall(boolean selfish, Color color)
        {
         Ball b = new Ball(canvas, color , ballid);
                canvas.add(b);
                BallThread thread = new BallThread(b, selfish);
                thread.start();
                ballid ++ ;
        }
public static ArrayList postsx = new ArrayList ();
public static ArrayList postsx = new ArrayList ();
```

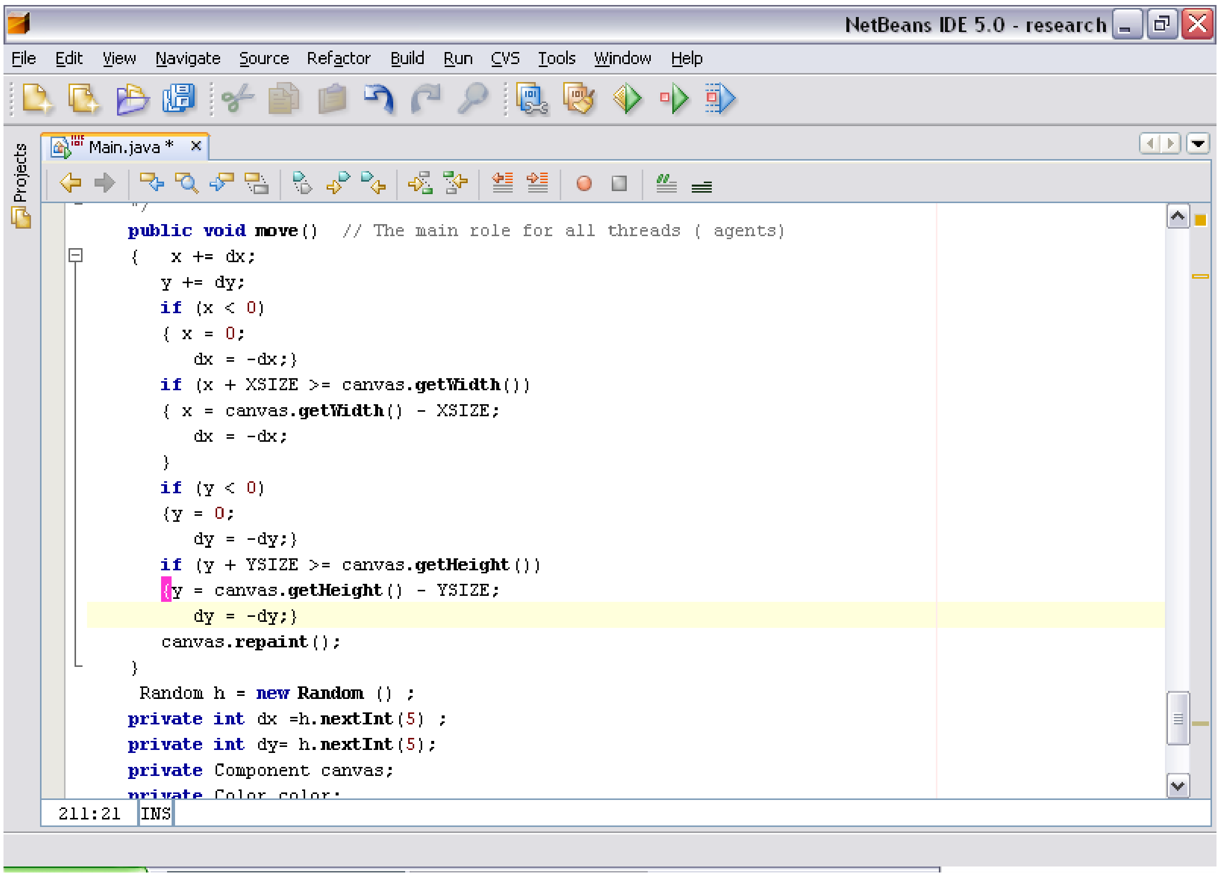

NetBeans IDE 5.0 - research
File Edit View Navigate Source Refactor Build Run CVS Tools Window Help
Projects
Main.java *
97:6
INS

```
class BallThread extends Thread
{/**
      Constructs the thread. @aBall the ball to bounce @boolean selfishFlag true if the thread
  * is selfish, using a busy wait instead of calling sleep*/
   public BallThread(Ball aBall, boolean selfishFlag)
   { b = aBall;
      selfish = selfishFlag;}
   public void run()
   {try
      {  for (int i = 1; i <= 100000; i++)
         { b.move();
            if (!selfish)
            {// busy wait for 5 milliseconds
               long t = System.currentTimeMillis();
               while (System.currentTimeMillis() < t + 5);
            }else
               sleep(5); }
      }
      catch (InterruptedException exception)
      {
      }
   }
```

Figure 11: Part of code in JAVA presenting the behavior of agent ball class

We see in Figure 11, JAVA programming class () presents a simple agent (having full agency characteristics) that performs simple role. The intelligent factors in this agent class are:

i. Smart balls pushed into environment using buttons start or selfish (see Figure 7) they start into runnable threads
ii. Check if the smart ball (agent) is normal or selfish
iii. Selfish agent has priority to move in the frame and push less priority agents
iv. Selfish is moving with faster pace than normal agent
v. Agents (balls) are performing moving role see Figure 12 this role has some rules the agent must consider:
vi. Each ball will move by a random (from 0 to 5) distance to random direction in the frame
vii. Each agents (selfish or normal) will not be allowed move out the frame


NetBeans IDE 5.0 - research
File Edit View Navigate Source Refactor Build Run CVS Tools Window Help
Projects
Main.java *

```
 */
public void move()  // The main role for all threads ( agents)
{   x += dx;
   y += dy;
   if (x < 0)
   { x = 0;
      dx = -dx;}
   if (x + XSIZE >= canvas.getWidth())
   { x = canvas.getWidth() - XSIZE;
      dx = -dx;
   }
   if (y < 0)
   {y = 0;
      dy = -dy;}
   if (y + YSIZE >= canvas.getHeight())
   {y = canvas.getHeight() - YSIZE;
      dy = -dy;}
   canvas.repaint();
}
 Random h = new Random () ;
private int dx =h.nextInt(5) ;
private int dy= h.nextInt(5);
private Component canvas;
private Color color;
```

```
211:21  INS
```

Figure 12: Part of code in JAVA of moving role class that agent ball can lock with

Another example of applying multi agent system using ARL is shown in Figure 13 , that represents an environment of cars in corridor that leads to number of rooms, each car objective is get into a room , with no more one car in a room as soon as a care leave a room (by internal intention) to serve another objective any closest car will fill that room , however, because the number car is greater that the number of rooms , the rest of cars will keep moving until a room is available, (*agents)* that can lock to roles moving in four direction (forward , backwards, right , left) , the agents which represents in car, the entire car system and its coordination is considered MAS.

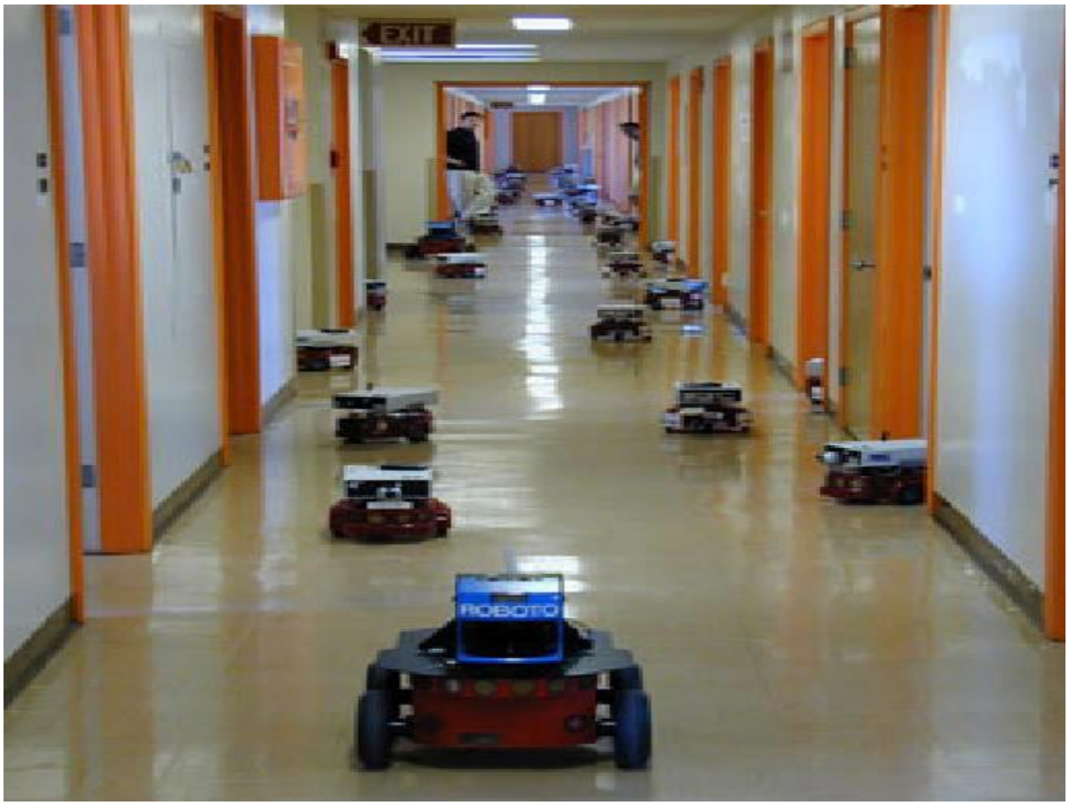


Figure 13 : Real environment that represents smart cars allocated in rooms and moving in corridor

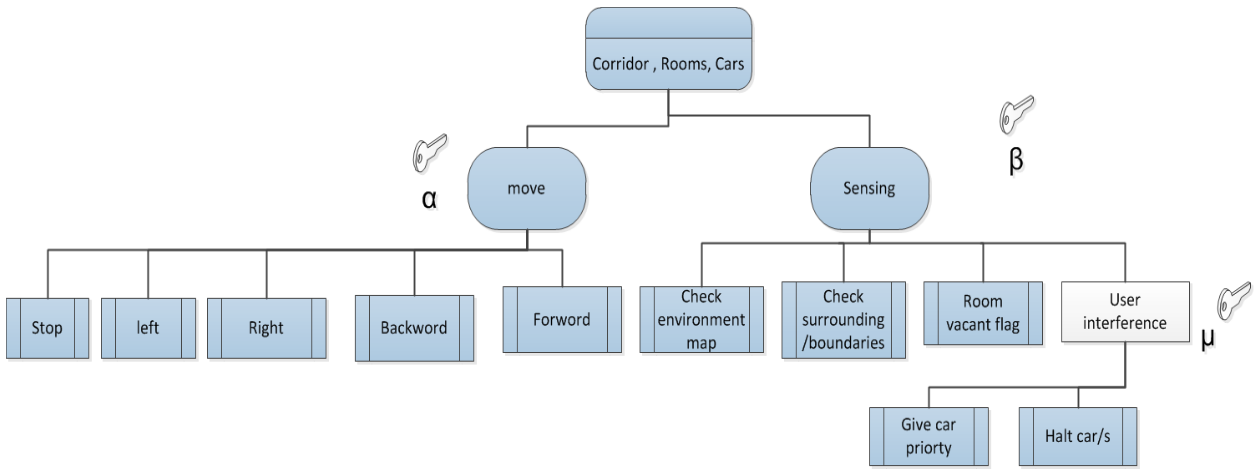


Figure 14: Organizational View of smart cars in corridors

Figure 14 depicts the organizational view of autonomous smart cars moving in corridor to occupy one room for each car. There are two *super roles*, firstly, *moving* ,agents must have *(α key)* to perform atomic roles with this super role on cars , these roles concerns moving cars in all directions or stop. Another *super role*, sensing with (*β key*) that responsible to sense and communicate with surrounding environment that makes sure to move in right track not to hit another car or walls and to check which room is available to go in, or go out of room according to internal state or outside *user interference*, which represents a sub role to halt all system or interfere to give certain car more advantage to speed up to allocate a room.

As shown in previous example, we can apply the smart cars and rooms systems in Java threads, each car has agent, this agent can lock and unlock into various atomic roles that defined in Figure 15. This article has defined integrated view of engineering and application view of Smart agent system. Nevertheless, when make this system run there is no grantee which thread will starts first, on this base there would be no clear path or preplanned scenario of system run. The classical view of *main ()* to start system in object oriented system is now over. in agent base system each thread (Agent) has many candidate *main ()* to lock with and to start.

4. CONCLUSION

This article presents new computing paradigm has its engineering and programming views. It based on smart *Agent Orientation or Agent Based* software, provides two examples of using smart agents in real system , first is concerning with software system and the second is concerning with smart robotic system . The main value of this article is to give an overall view of a simple real environment of Multi Agent System MAS supported by application proof, and to show the integration and incorporation of modeling tools and programming techniques in multi agent systems. The idea is to solve complexity of MAS through decomposition. The complexity of multi agent system is managed by breaking it into roles and agent classes, basically, roles are considered as *positions* and agents are considered as *people* fill those positions, within this article we will attest a look on programming codes of agents and roles classes to represent a new agent programming paradigm, we notice that those agents and roles classes codes are in fact reflecting agent engineering methods and techniques based on Agent Role Locking ARL theory.

For the sake of code object oriented language -Java IEEE- has been used as an application of this theory., accordingly we use this view for applying *smart bouncing ball* as software system , and *smart cars* in corridor as robots system..

The program execution *run* fore agent orientation case will be totally different from the case of *main () run* of object orientation as , in the first case all threads of agents released in run time memory and start running (activated) or stop (deactivate / sleep) according to agent internal state and intentions to satisfy agent base system main objectives. For the case of smart agent threads no guarantee which threads will start first because it depends on system functionality or depends on agent internal state (*goal*), with a result of there is no guarantee or preplanned scenarios if how agents will act after they released in memory in the runtime.

REFERENCES

[1] Hang-Jiang, Q. Zheng, L. Lei, S. Li-Ping and H. Xing-Chen. Formal Specification and Proof of Multi-Agent Applications Using Event B. Information Technology Journal. Vol. 6 issue 8, Pp. 1181-1189. 2007

[2] Z. Hou, Z. Yu, W. Zheng and X. Zuo. Research on Distributed Intrusion Detection System Based on Mobile Agent. Journal of Computers, Vol. 7, No. 8, pp. 1919-1926. August 2012.

"doi:10.4304/jcp.7.8.1919-1926".http://dx.doi.org/10.4304/jcp.7.8.1919-1926

[3] William Rand; Uri Wilensky (November 15–8, 2007). "Visualization Tools for Agent-Based Modeling in NetLogo". Agent2007. Chicago, IL. Retrieved October 4, 2012.

[4] Andreas Viktor Hess, Øyvind Grønland Wollerm. PhD Thesis, Multi-Agent Systems andAgent-Oriented Programming ,DTU Compute, Technical University of Denmark Matematiktorvet, Building 303B, DK-2800 Kongens Lyngby, Denmark, Lyngby, 01-July-2013

[5] Ingrid Nunes, Elder Cirilo, Carlos J. P. de Lucena, Jan Sudeikat, Christian Hahn, Jorge J. Gomez-Sanz. Agent-Oriented Software Engineering X , Lecture Notes in Computer A Survey on the Implementation of Agent Oriented Specifications Science Volume 6038, 2011, pp 169-89 ISBN 978-3-642-19207-4

[6] Michael Köster, Federico Schlesinger, Jürgen Dix 10th International Workshop, ProMAS 2012, Valencia, Spain, June 5, 2012, Revised Selected Papers

[7] Salaheddin Juma Juneidi and G.A.Vouros. Survey & Evaluation of Agent Oriented Software Engineering main approaches . International Journal of Modelling and Simulation 01/2010; book(2010):1-15. DOI:10.2316/Journal.205.2010.1.205-4306

[8] L. Jemni Ben Ayed, and F. Siala. Specification and Verification of Multi-Agent Systems Interaction Protocols using a Combination of AUML and Event B. XVthInternational Workshop on the Design, Verification and Specification of Interactive Systems DSV-IS 2008, LNCS 5136, Kingston, Ontario, Canada, pp. 102-107. 2008

[9] Marc-Philippe Huget, Extending Agent UML Sequence Diagram, F. Giunchiglia ct al. (Eds.) : AOSE 2002 ,Lncs 2585, pp 150-161, Springer-Verlag Berlin Heidelberg 2003.

[10] Salahededdin J. Juneidi, George A. Vouros "Agent Role Locking (ARL): Theory for Agent Oriented Software Engineering" , IASTED International confrence SE November 9-11, 2004, MIT, Cambridge, MA. USA.

[11] Michael Wooldridge , Nicholas Jennings , David Kinny , Autonomous Agent and Multi- Agent System , 3, 285-312,2000 : The Gaia Methodology for Agent-Oriented Analysis and Design. 2000 Kluwer Academic Publisher 20