ABSTRACT
Agent orientation introduced as a new paradigm of computing paradigm that calls for new approaches in both of software engineering and programming methodologies. In the past two-decade we attested many ideas and methodologies to support agent orientation. Throughout agent related research history, there was a clear gap between agent software engineering approaches and applications to support such approaches through programming. This paper will tackle the application view over agent oriented software system, (Agent Role Locking) ARL theory is used to design and implement agents in software system. The main aim of this paper is to show innovative incorporation relationship between engineering methodologies and programming application in agent orientation technology.

Key Words
Agent Role Locking Theory (ARL), Java Programming, Agent class, Role Class.

1. INTRODUCTION
1.1 Objects and agents: Bottom lines
Several comparisons took place between agents and objects as software entities to have clear view toward objects and agents, actually, most of these differentiations give no absolute bottom lines between objects and agents in terms of definition and application as mentioned above. It is worthy to have these bottom lines between objects and agents to have a final decision when and where agents are embedded in our software, the most we concern bout is the minimum requirements needed for this intelligent software entities to be considered as an agent. In [12, 13] has discussed three requirements:

- **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 at what time.

- **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.

- **Proactivity.** In order to accomplish its design 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 agent base specific 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. new kinds of fifth generation languages, agent-base libraries and agent tools have appeared to represent agent-based modeling that are useful for gaining understanding across disciplines and fields: there are a growing number of agent-based applications in a variety of fields. Agent-based modeling can be used to study of dynamic and unpredictable phenomena as consumer choice others are social network analysis, network science, GIS, the spread of epidemics, and behavioral economics in stock market analyses, for example, and it can also be used to understand how people's behavior affects activities like pedestrian movement, transportation traffic. And machine learning.

![Figure 1: proposed Agent Base Modeling Software Development](image-url)
1.2.1 Jack:

Jack is an agent development IDE that extends Java (another example likewise is JADE) to incorporate the concepts of agent-oriented programming. So in place of objects, under AOSE, it talks about agents. From an implementation perspective, these agents having the same fundamental members as in objects (data members, functionalities), have additionally the following constituents [2]

**Capabilities**: is the name given to the reusable components of agents, just like the modules in object-orientation. They encapsulate the reasoning constituents (events, plans, sub-capabilities, etc.) to provide a certain ability to any agent.

**Plans** are similar to functions in object-oriented classes. They are the instructions the agent follows to try to achieve its goals and handle its designated events.

**Events** trigger plans. Just like we have event handlers in .NET, we have plans in Jack. And they are executed as soon as certain events occur.

**Belief Sets** represent agent beliefs using a generic relational model. Queries can be applied on them and, when some changes occur, events can be associated with those changes.

![Figure 2 : Project main components in Jack](image)

1.2.2 NetLogo:

NetLogo is programming language developed for agent-based modeling. It has several built-in tools for running experiments on agent-based models and collecting data. One is BehaviorSpace, allowing optimization of parameter settings—such as decreasing transit time to the minimum amount possible. Another, HubNet, allows for participatory simulations in scenarios where humans are involved in decision making. A GIS (Graphic Information System) extension allows for high-performance computing clusters and even simulation of cities[3].

**Emerging Order**

Agents are decentralized, suggesting there is no central authority. They have behaviors and they operate according to what those behaviors dictate they should do. Although agents are autonomous and act independently, they also interact with other agents. This does not mean every agent in a simulation interacts with every other agent all the time, but their environment is dynamic and local.

**Agent Features**

An agent-based model consists of four features: a set of agents defined by the user; a set of agent relationships, also defined by the user; an environment in which the agents live, such as infrastructure and roads; and a framework for simulating all these agent behaviors and interactions. Agent-based modeling begins and ends with the agent perspective, such as how a driver would behave in different situations. These will perceive the following five essential features for an agent to have: self-contained; autonomous or self-directed; senses information and translates it into actions; has a state that varies over time; and it is social and has dynamic interactions with other agents.

![Figure 3 : NetLogo model to represent sheep and wolf agents in grassy environment](image)
There are other features often seen in agents, but they are not essential to a good model.

**Model Scenarios**

There are different typologies in terms of how agents interact spatially, and agent-based models can often look different. People are working on larger scales now, using realistic maps and Geospatial Information Systems (GIS), and agents are connected through networks. According to Macal, most of the original agent-based models, including traffic models, were based on cellular automata—a field developed by John von Neumann in the 1940s looking at self-replicating machines[3].

1.2.3 GOAL:

To design agent systems, programming using the paradigm of agent-oriented programming can be useful.

The key idea of agent-oriented programming is that agents are programmed in terms of mentalistic notions (such as belief, desire, intention) that represent the properties of agents[8,9]. The design of a program is therefore centered around designing intelligent agents that have the characteristics defined in the previous section. Primarily that the GOAL is an agent programming language for programming rational agents. GOAL agents derive their choice of action from their beliefs and goals. The language provides the basic building blocks to design and implement rational agents. The language element sand features of GOAL allow and facilitate the manipulation of an agent’s beliefs and goals and to structure its decision-making. The language provides an intuitive programming framework based on common sense notions and basic practical reasoning. agents are autonomous, reactive, proactive, and social.

The structure of GOAL programs [4].

![Figure 4: The structure of GOAL programs (from [MPI])](image)

2. AGENT ROLE LOCKING THEORY

ARL

ARL 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 recognized by the most general objectives of the system, within each organization, their are associated objectives and tasks, these objectives and tasks are performed by positions with responsibilities, like secretary as a position responsible on meeting arrangements, daily schedule checking, read and replay to e-mails, ...etc. figure 4 defines ARL decomposition in which reflects on Figure 5 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 5 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.

---

1 Going into details of ARL analysis and design of MAS is beyond this paper, consider references [17,18] to have an idea about ARL.
entities (responsibilities). This ARL view of MAS, as organizational view and role decompositions, is consistent with role definition in [17]:

- **Objectives**: is represented in ARL by atomic role entity.
- **Responsibilities**: ARL specifies agent class entity as responsible to perform roles.

**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 static structure) that can be instantiated. Agent in MAS are not equal, they are different according to their goals, permissions entitle the agent for 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.

**Figure 5**: An overview of ARL on MAS environment decompositions and role definition

### 2.1 Dynamic Model

Which involves in collecting interdependent atomic roles presented into Atomic Roles Couples table (ARC) (see ARL references [17,18]), this table defines the couples of interdependent roles and their candidates performing agents. **Interdependent role** contains dependant 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 \( \alpha \) key, for the Y super role the satisfying agent must own \( \beta \) key. To specify the details of interaction between roles couples, the Agent Interaction Protocols (AIP) is used.

AIP firstly presented [16,17]. 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 5 of AIP for two interdependent roles X, Y and the agents to perform.
2.2 Static Model

ARL proposes new development on static structure of Agent-Class 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 new static view development could be integrated with UML reaching to AUML [11,17] that supports object class, as well as agent class and role class, to represent static models of MAS. 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.
2.3 Definitions and Assumptions in ARL

- An agent instant of agent type is independent from role entity. Agent entity is idle and activated when locked into role using given key.
- Running agent-role (and therefore, for the system to run) an agent must launch or satisfy a role.
- Each agent class represents a unique agent type.
- 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.
- 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.
- 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.
- 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.
- 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.
- 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.
- 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.
- An agent instance performs one and only one atomic role at a given time.
- Agent - role switching constrains: An agent may unlock from one role to lock into another with respect to the following constrains :
  - Passive Sensing: Performing unlocking and locking according functional events and constant time interval.
  - 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.
  - 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.

ARL is contributing to increase the acceptance and the practical usability of AOSE as a new computing paradigm. As it has overall view of MAS dynamic and static structure, presenting various modeling tools for agents and roles, as we going to see in the next section an application of agent systems using ARL on object programming language.

3. ARL APPLICATION

ARL provides new look for MAS by separating agent classes and the role classes “new look: breaking complexity through decomposition” - which make it possible to control agents and roles that may eligible to perform. Java is object oriented programming language that will be used to program agents and roles classes. Most important the characteristics of agent that mentioned previously are conserved (autonomous, proactive, mobile).

We going to use simple example (Bouncing Ball) that will show diversity of agents behavior - which basically is throwing balls in 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
given to the agents are 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 5 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 theirs periodic circumstances without any forced or pre-planed schedule and even without central or external interfere or control on this example we have two types of agents, and we have only one super role which is Move that has three atomic roles: check_next_position(), do_move(), and wait_time(). Figure 7 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.

3.1 Agent and Threads in Java
In the Java programming language, threads can be used in this application to represent agents, 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.

3.2 Thread Properties
3.2.1 New threads
When 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 it is in the new state of runtime in 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, 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 are represented of JAVA threads and the positions of these balls are stored in array list objects as the following code:

```java
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 postsy = new ArrayList();
```

The presented below software is JAVA programming class (Figure 11) represents a simple agent (having full agency characteristics) that performs simple role. The intelligent factors in this agent class are:

- Selfish agent has priority to move in the frame and push less priority agents
- Selfish is moving with faster pace than normal agent

Agents (balls) are performing moving role see Figure 12 this role has some rules the agent must consider:

- Each ball will move by a random (from 0 to 5) distance to random direction in the frame
- Each agents (selfish or normal) will not be allowed move out the frame

```java
figure 11. agents (Threads) in Java language runtime
```

```java
class BallThread extends Thread
{
    //Ball the ball to bounce boolean selfishFlag true if the thread
    //as selfish, moving a busy wait instead of calling sleep()
    public BallThread(Ball ball, boolean selfishFlag)
    {
        b = ball;
        selfish = selfishFlag;
        public void run()
    }
    try
    {
        for (int i = 0; i <= 100000; i++)
        { b.move();
        if (!selfish)
            { // busy wait for 1 millisecond
                long t = System.currentTimeMillis();
                while (bSystem.currentTimeMillis() < t + 5);
            }
            else
                sleep(50); }
    }
    catch (InterruptedException exception)
    {
    }
}
```

```
figure 12: Part of code in JAVA presenting the behavior of agent ball class
```
4. CONCLUSION

This paper presents new computing paradigm in terms programming and Engineering that is known as Agent Orientation software, and provides an overall view of simple real environment of MAS supported with application proof. The main aim of this paper is to show importance of integration and incorporation of modeling tools and programming techniques in multi agent system. The idea of solving complexity through decomposition is applied, the complexity of multi agent system is managed by breaking ABS into roles and agent classes. The reader has seen the first 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.

Actually object oriented language -Java IEEE- has been used as an application of this theory. ARL has full view and assumption for engineering Multi Agent Systems (MAS), accordingly we use this view to apply - bring to life - a very primitive agent bouncing ball.

by organizing the agents types and candidate roles they might play (lock with), and then release them in computer memory (RAM) with little intelligence factors, that enable them to move within MAS environment and run (activated) according to agent role locking constraints. This article is differentiating between agents to prove that we can make agent much smarter (Selfish) than other. In the history of ABS programming, listed above some other agent programming languages proposed for programming an agent, but this paper establishes for a new era of agent base computing by providing an agent an integration of engineering techniques and programming tools towards agent orientation

5. REFERENCES


