Full Review from Engineering to Programming: Smart Agents Application using ARL

By Salaheddin J. Juneidi

Palestine Technical University Khadoori

Abstract- Agent orientation has introduced as a new computing paradigm that calls for new approaches in both software engineering and programming orientation. In the past two-decade, we attested many ideas and methodologies to support agent orientation. Throughout agent-related research history, there was uncertainty how to apply smart agent in a software system, we have attested overwhelming theoretical approaches of agent software engineering, however, programming languages and orientation for this approach are still not well defined. 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 Smart Agent Orientated Technology.

Keywords: agent role locking theory (ARL), java threads, agent class, role class. agent-oriented programming AOP.

GJCST-C Classification: I.5.m

Strictly as per the compliance and regulations of:

© 2018. Salaheddin J. Juneidi. This is a research/review paper, distributed under the terms of the Creative Commons Attribution-Noncommercial 3.0 Unported License http://creativecommons.org/licenses/by-nc/3.0/), permitting all non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.
Full Review from Engineering to Programming: Smart Agents Application using ARL
Salaheddin J. Juneidi

Abstract: Agent orientation has introduced as a new computing paradigm that calls for new approaches in both software engineering and programming orientation. In the past two-decade, we attested many ideas and methodologies to support agent orientation. Throughout agent-related research history, there was uncertainty how to apply smart agent in a software system, we have attested overwhelming theoretical approaches of agent software engineering, however, programming languages and orientation for this approach are still not well defined. 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 Smart Agent Orientated Technology.

Keywords: agent role locking theory (ARL), java threads, agent class, role class. agent-oriented programming AOP.

I. Introduction

a) 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

b) 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 decades 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; another 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.
II. 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 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..n organizations, each organization is decomposed into various number of super role which represents a position with its specified authority (permissions) which represented in Figure 2 as a key, only agents which have certain keys can lock 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 static structure) that can be instantiated. Agents 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 the summary of those two models.
Figure 2: An overview of ARL on MAS environment decompositions and role definition

a) Super & Atomic roles

ARL emphasizes 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.

Figure 3: General organizational view of MAS with agents and keys
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 role of studying at university, (figure 4) agents are autonomous and can change roles at any time or according to their 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.

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

**b) Agent definition**

Each agent is given authorization and eligibility to play atomic roles, this eligibility is presented by super role assigned a 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 that for super roles in Figure 3 we have three agent types Administrator, student, and teacher, each of which has the right and permissions to perform the corresponding atomic roles, so these agent types have keys ($\lambda$, $\omega$, $\theta$ 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 lock though MAS environment, all that locking and unlocking take place according to MAS functionality specifications.

Returning 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 ($\omega$) key to able to lock into its appropriate atomic role.

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

i. **Role launching:** agent launches atomic role according to 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, the first column is launching atomic roles that need other roles to interact with, that’s 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 to internal state and functionality, so for a given unit of time we cannot guarantee a specific agent would be exist in an organization. In the next subsections, we are going through details of ARL definitions and assumptions.
c) **Static Model**

ARL proposes new development on the static structure of Agent-Class and Role-Class, and presents them as 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 view of new static model development could be integrated with UML reaching for 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 can instantiate according to 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 to 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.

![Table 1: ARC table, some selected interdependent atomic roles](chart.png)

**Figure 5:** Agent class (left) Role class right see [9]
d) 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 accomplished 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 starts 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].

![Simple agent role AIP diagram](image)

**Figure 6:** Simple agent role AIP diagram

e) 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, are 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 is 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 a 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 an overall view of MAS dynamic and static structures, presenting various modeling tools for agents and roles, as we are going to see in the coming sections, attesting an application of agent systems using ARL on multithreads programming language.

III. 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 have 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](image)

### a) 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 are 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 a 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 Ball Thread 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.
b) **Thread Properties**

i. **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 runtime OF 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 does nothing until new state and circumstances applied to start execution.

ii. **Runnable threads**

Once the agent locked into a 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, that is how multi-threading is applied in Java program runtime.

c) **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:
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();
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 the 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 delay movement of 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.

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

IV. Conclusion

This article presents new computing paradigm in terms programming and Engineering based on Agent Orientation or Agent Based software, and provides an overall view of a simple real environment of MAS supported by application proof. The main aim of this article is to show importance of 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, we will attest 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 for applying smart bouncing ball as software system, and smart cars in corridor as robots system.

Must be mentioned in this article that, the program execution run fore agent orientation case will be totally different from the case of main() run of object orientation as, in the second 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. So that means no preplanned scenarios of which agent is active or which is inactive.

REFERENCES Références Referencias