***APPLICATION OF RICART-AGRAWALA FOR SYNCHRONISATION OF 3 SITES WHICH WANT TO ENTER CRITICAL SECTION***

# About The Ricard-Agrawala algorithm

**Ricart–Agrawala algorithm** is an algorithm to for mutual exclusion in a distributed system proposed by Glenn Ricart and Ashok Agrawala. This algorithm is an extension and optimization of Lamport’s Distributed Mutual Exclusion Algorithm. Like Lamport’s Algorithm, it also follows permission-based approach to ensure mutual exclusion.[1]

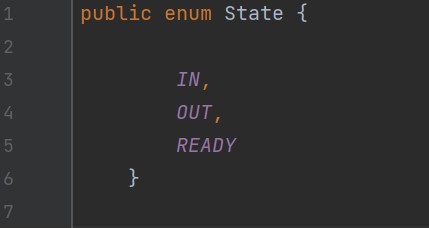
In this algorithme:

* Two type of messages ( **REQUEST** and **REPLY**) are used and communication channels are assumed to follow FIFO order.
* A site send a **REQUEST** message to all other site to get their permission to enter critical section.
* A site send a **REPLY** message to other site to give its permission to enter the critical section.
* **Requesting site enters its critical section only after receiving all reply messages**. Upon exiting the critical section, the site sends all deferred reply messages.
* A timestamp is given to each critical section request using Lamport’s logical clock.
* Timestamp is used to determine priority of critical section requests. Smaller timestamp gets high priority over larger timestamp. The execution of critical section request is always in the order of their timestamp.

The code has 5 classes namely:

* Node class which is the mother class that describes a node and its behaviors
* NodeOne, NodeTwo and NodeThree are the 3 sites (Agents) that inherit from the Node class
* State is an enumeration which contains the set of states that an agent can have.

**The State enumeration**

***fig:enum***

According to the Java naming conventions, we should have all constants in capital letters. So, we have enum constants in capital letters.

Java Enums can be thought of as classes which have a fixed set of constants.[2]

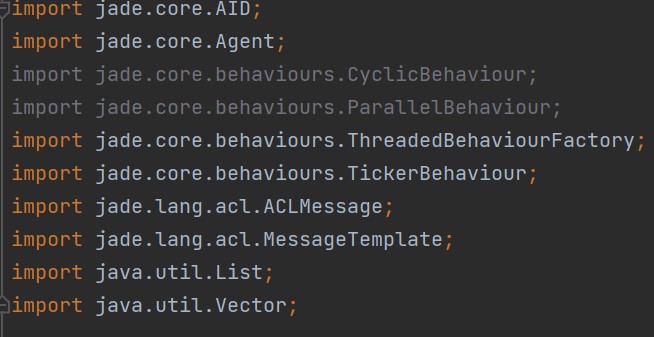
As defined in the Ricard Agrawala algorithm, a site can either be in the following states

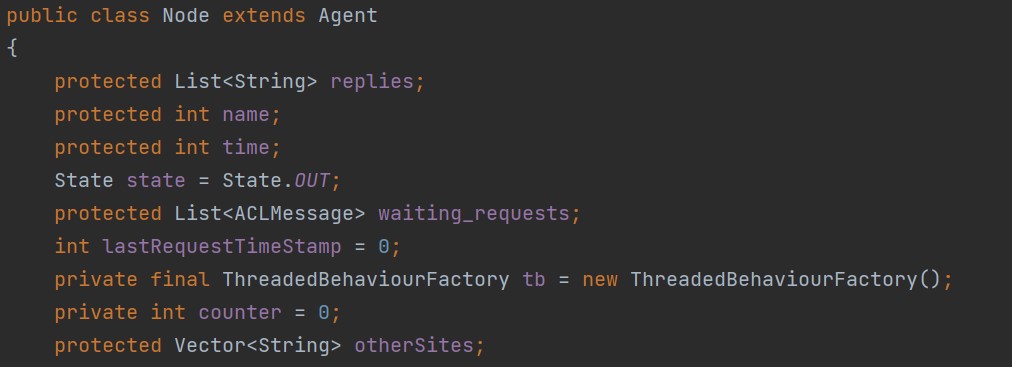
***READY*** - thus it wants to enter critical section

***In*** - thus it’s in the critical section

***Out*** - state meaning it is out of the critical section and is not even asking to enter yet.

**THE NODE CLASS**

fig: imports



# fig:variable declarations

All protected declarations are so that the global variables are accessible from the rest of the other sites.

Initially the states of all sites is *OUT.*

List<String> replies;

Is a list that contains replies from other sites.

int name;

Contains the site number

By default, 1 agent = 1 thread.

But you I decided to let behaviors live in their own threads.

Then when adding your behavior to agents, I only need to **wrap()** it through the use of a *ThreadedBehaviourFactory().* This allows my behaviors to run concurrently.

And the behaviors are added as shown below :



# fig: Adding behaviors

I used the Lamport class which extends *TickerBehaviour* such that it periodically executes a defined piece of code.[3]

Its purpose is that onTick, it just increments the value of the clock.

The time interval is defined for each site.

I also used the Server class which receives requests and sends acknowledgements .

For the server class, the action performed upon every tick are such that:

* + It wakes up upon receipt of a request.
  + It reads the clock time and the site number from the received message with the help of a function split which separates the received string where it will meet the comma (,) as shown below .

|  |  |  |  |
| --- | --- | --- | --- |
| • | int timeStamp = Integer.*parseInt*(request.getContent().split(" | | ,")[0]); |
|  | int siteName = Integer.*parseInt*(request.getContent().split(" | ,")[1]); | |

* + it then updates the clock by comparing the received timestamp and that of its own.

|  |  |
| --- | --- |
| • | counter = Math.*max*(counter, timeStamp) + 1; |

The site will then determine its response by considering :

* if the site itself is in state Out, it immediately sends the reply.
* If it’s in state ready, it then compares their clocks.
  + If the requesting site has lower timestamp, it the send it a reply
  + If its timestamp is higher, it adds the request in the queue.
  + Otherwise, if they have the same time clock, we consider their site numbers.
  + The site with the lower number gets the privilege.
  + Finally, if the request is received whilst it’s in the critical section, it blocks the process and puts it in the waiting list.

***fig: request***

# CLIENT CLASS

The client sends request and receives acknowledgements.

It sets its state to ready, and then in a loop the size of the number of sites available,

It will send a request message.

for(String site : otherSites)

request.addReceiver(new AID(site, AID.*ISLOCALNAME*));

lastRequestTimeStamp = counter;

request.setContent(lastRequestTimeStamp + "," + name); send(request);

The agent will then be ready to receive requests from other two sites.

It will then compare the timestamps and increment the value by one.

After receiving the 2 replies, it then enters the critical section.

counter = Math.*max*(counter, Integer.*parseInt*(msg1.getContent())) + 1; // modify the time stamp

Print();

state = State.*IN*;

## Printing the happenings

StringBuilder in Java is a class used to create a mutable, or in other words, a modifiable succession of characters. I used it to print out the replies obtained so far by a site. All other details are also input by the printing class.

|  |
| --- |
| private void Print()  {  StringBuilder build = new StringBuilder();  build.append("------------").append("Site ").append(name).append("-----  --------").append("\n");  build.append("Time Stamp : ").append(counter).append("\n"); build.append("State : ").append(state).append("\n"); if(state == State.*READY*) build.append("Req\_").append(name).append(" Timestamp :  ").append(lastRequestTimeStamp).append("\n");  build.append("Replies :").append("{ ").append(PrintReplies()).append("  }").append("\n"); build.append("Queued requests :  ").append("{").append(PrintRequests()).append(" }").append("\n"); if(state == State.*IN*)  build.append("Site ").append(name).append(" enters its critical section").append("\n");  build.append("-------------------------------------------------------------------");  String message = build.toString();  System.*out*.println(message); } |

***fig: Printing class***

## Node class

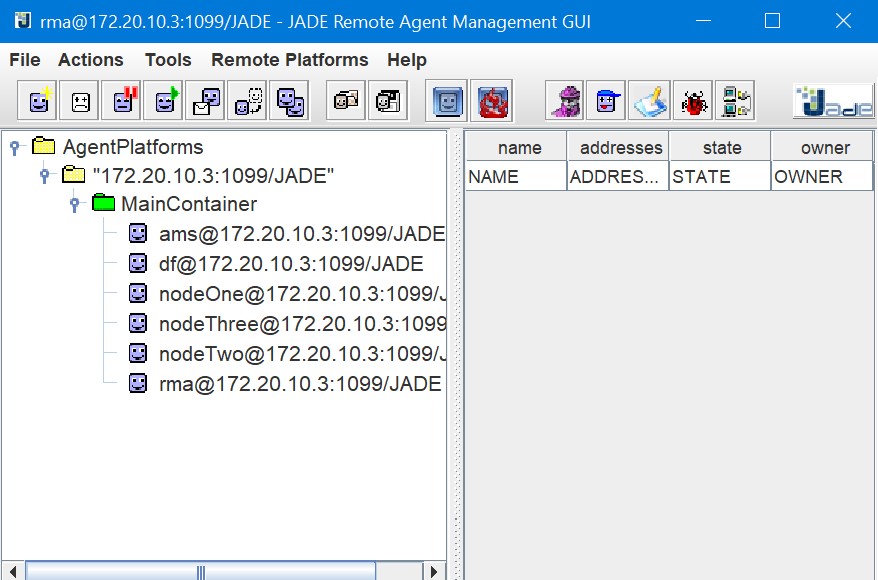
Each node class has similar code but just personalized to itself. And they all inherit from the Node class.

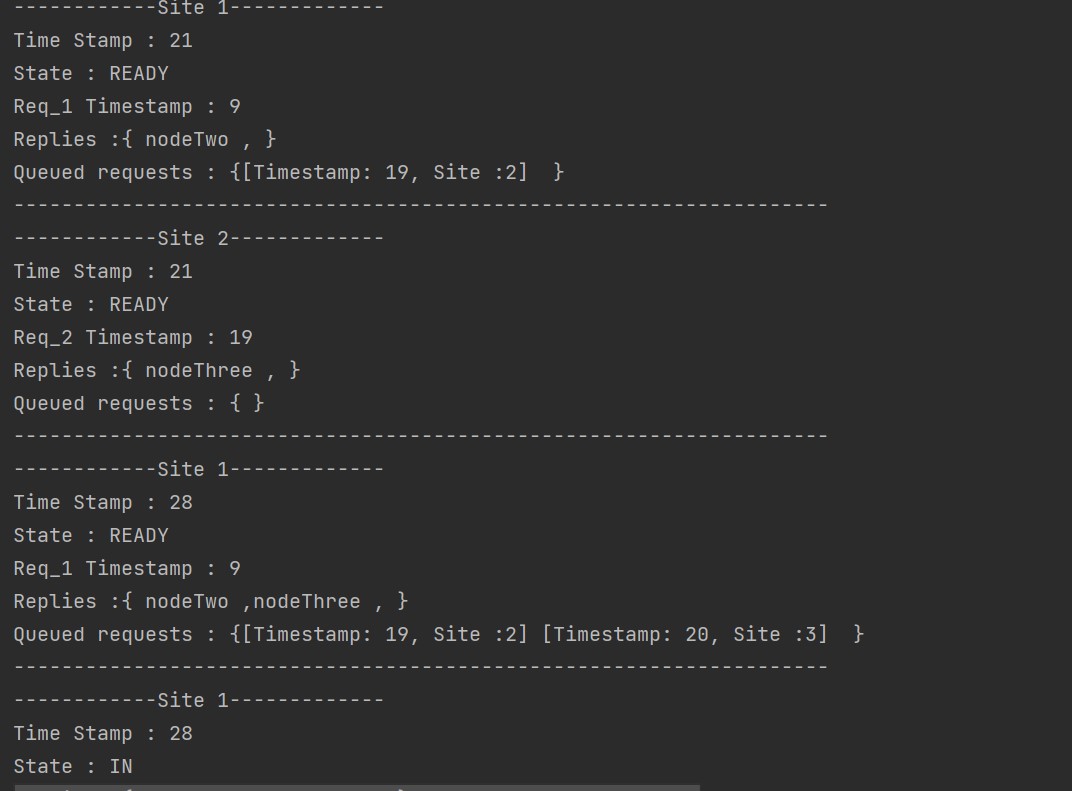
Each site has:

* A name
* A waiting request list
* A reply vector which it has received
* An increment interval of its clock

The Main class sets up the main container, adds the 3 agents and stars them and sets the Jade GUI to visible.

# Execution

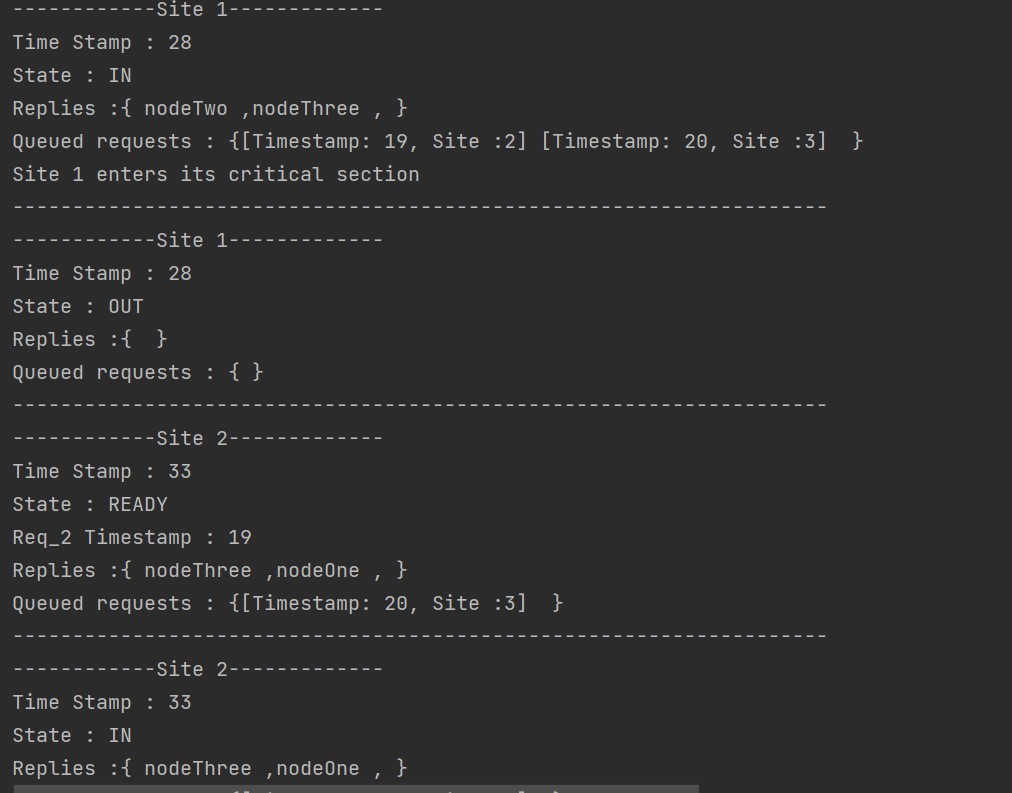




**fig:Execution1**

**Fig :execution**

**2**



# References

1. [https://www.geeksforgeeks.org/ricart-agrawala-algorithm-in-mutual-exclusion-indistributed-system/](https://www.geeksforgeeks.org/ricart-agrawala-algorithm-in-mutual-exclusion-in-distributed-system/) date visited 23 May 2022
2. <https://www.javatpoint.com/enum-in-java>
3. [https://www.unimuenster.de/Informatik/u/lammers/EDU/ws03/Landminen/Abgaben/Gruppe4b/jad e/doc/api/jade/core/behaviours/TickerBehaviour.html](https://www.uni-muenster.de/Informatik/u/lammers/EDU/ws03/Landminen/Abgaben/Gruppe4b/jade/doc/api/jade/core/behaviours/TickerBehaviour.html)