UML Modeling of a Protocol for Establishing Mutual Exclusion in Distributed Computer System

Dr. Vipin Saxena[†] **and Deepak Arora**[††],

Department of Computer Science, B.B. Ambedkar University (A Central University), Vidya Vihar Rae Bareli Road, Lucknow U.P. 226025, India

Summary
Mutual Exclusion in a fully distributed computer system contains a lot of complexities related to the solution design and its implementation. A number of solutions have been proposed for solving mutual exclusion problem for fully distributed environment but most of them are related to higher message passing overheads over the network, which causes low performance and efficiency. In this paper authors have presented a design of a protocol for establishing mutual exclusion among processes running concurrently, in fully distributed environment by the use of bidirectional ring technique. A well known modeling approach i.e. Unified Modeling Language (UML) is used to model the protocol. UML class diagram, sequence diagrams related to various scenarios is designed and system complexities are also measured.

Key words:
UML Class Diagram, Bi-Directional Ring, Distributed system, Mutual Exclusion, UML Sequence Diagram.

1. Introduction

Distributed computing systems have become more popular nowadays, due to its capability to deal with heterogeneous environments, network links with varying latencies, and unpredictable failures in the network or the computers. In this case mutual coordination becomes more important aspect, when measuring the performance and reliability of any distributed computer system. Also the mutual coordination to access any shared resource should be synchronized in such a way that at a time only one process can access it at a time. Each process contains a segment of code, in which it can update the shared variables or resource [1]. For accessing its critical section, a process has to submit a request for accessing the resources and after the completion of critical section execution it should release it, so that other processes can execute their own critical sections. The necessary conditions for any mutual exclusion algorithm are given below [1, 2],

1. Only one process can execute its critical section at a time.
2. Progress must ensure, if no process is executing its critical section, and some processes want to enter in their critical section then permission will be given to only those, who have not been entered yet in their critical section, in a finite time.
3. Entry of any requesting process in its critical section must be finite time bounded.

A number of researchers have given solutions to solve the mutual exclusion problem in a distributed environment but most of them are related to heavy message passing. Lamport, L [3] suggests that any process wants to enter in its critical section will issue a message and sends it to all other existing processes. When the issued process gets back the messages from all of other processes then only it can enter in its critical section. After finishing its critical section execution, it will again send message to all other processes in the system. In this case for N process, there will 3(N-1) messages are required to fulfill on process request. According to Ricart, G., Agrawala, A. [4], there is no need to send release message. In this way it requires only 2(N-1) messages to fulfill a process request. Maekawa, M. [5], has solved the mutual exclusion problem by using sets. Any process can execute its critical section, when it receives permissions from all other processes its set. It requires 3√N massages to fulfill a process request. Agrawal, D., El Abbadi, A. [6], suggested an approach based on token ring. A process can enter in its critical section, if it holds the token. In this case N/2 messages are required for handling a process request. Suzuki, I., Kasami, T. [7], discussed the solution as, the process wants to enter in its critical section then it will send messages to all and the processes, which currently hold the token and then sends the token to requester process. It requires N massages to fulfill a process request. Rymond, K. [8], suggests an approach based on tree, in which, root always holds the token. If any process wants to access its critical section submits its request to its parent node and this request will propagate towards root and token will send to the requester child, if it is on the top of request queue. It requires 2logN messages to fulfill a process request. Another important algorithm proposed by, Kawsar, F., Shaikot, S.H., Saikat, S., Mottalib, M.A. [9], uses token ring approach based on token's time stamp, request list and node_no. when token matches with the stored copy of token, a process can enter in to its critical section. It requires N messages to fulfill a process request.

Manuscript received June 5, 2008.
Manuscript revised June 20, 2008.

Chakraborty, R.N., Yaprak, E. [10], suggest that one can improve the reliability of token ring, if it is used with a bi-directional ring, also given an algorithm for the detection of ring breakage from single link component failure.
Lots of work on UML modeling is done by various researchers but a very few research papers are available on UML modeling related to the distributed computer systems developed by Booch [12], is the most suitable visual presentation platform for modeling the real word problems. OMG [13, 14] describe the latest UML specifications related to real world, modeling aspects also the way of representation for XML metadata specification in UML diagrams and the standard storage representations. Pllana and Fahringer [15] suggested UML profiles, through which one can model the high performance applications. Further one more important reference by Pllana and Fahringer [16], describes the issues related to the design of parallel and distributed applications, using Unified Modeling Language (UML). An object oriented distributed architecture system is recently defined by Saxena, V., Arora, D., Ahmad, S. [17], through UML modeling, along with a case study of a multiplex system for the distributed computing environment.
Most of the research work describe above is based unidirectional ring technology. In this paper authors have designed an algorithm for establishing mutual exclusion in a distributed environment, using token approach based on bi-directional ring technique, also satisfy the conditions mentioned above. The modeling of the designed protocol is given with the help of Unified Modeling Language (UML). Also authors have suggested a way to recover from token lost and recovery of bi-directional ring from the process failure, which are major hazards of token based approach. The performance of the designed algorithm is also reported in the form of total messages required for a process to enter in its critical section, along with the comparison of previous approaches.

2. Background

2.1 Mutual Exclusion in fully distributed system

Distributed computer system is nothing but an interconnection of loosely coupled processors through a communication network. Every processor is having its own local resources and intended to share the remote processors, resources, which may vary in size and function. The main purpose of distributed is to provide efficient and convenient environment for this type of resource sharing. However access of remote resources, imposes more overheads in terms of processing and network communication, but the development of distributed computer systems is continuing getting its new hike, cause of its reliability and high performance.

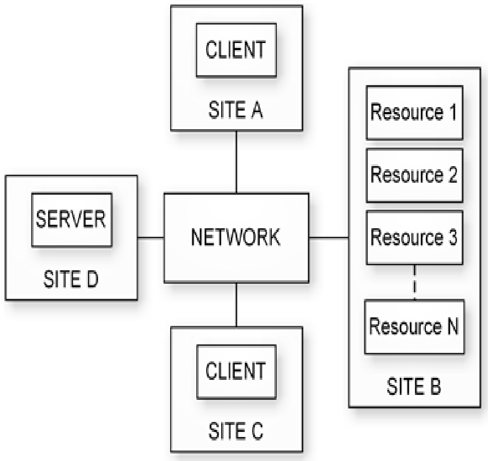

Fig.1 Sharing of resource under distributed computer system.

Distributed processes often need to establish coordination in their activities. In the group of processes, mutual exclusion for sharing of resources is required for preventing interference & provides consistency when accessing the shared resources. Therefore, there should be some method used to establishing the mutual exclusion for accessing the resources. Sharing of resources under the distributed computer system is shown in Fig. 1.
All the four sites namely A, B, C and D are interconnected through the communication network and these can share the local as well as remote resources. In the above figure, request can be generated at any time from any process to get the resources, which may be available locally as well as globally.
In this way more than one process can request for a common resource. Here the mutual coordination becomes important among the processes running concurrently in a distributed computer system.
Also one can say if there are $\{P_0, P_1, P_3 \ldots P_N\}$ process in the system then each process must have a segment of code, called critical section, in which a process may access the resource or any common variable.
The most important aspect here is when process is in the critical or executing critical section then no other process is permitted to enter in its critical section. Thus the execution of critical section by process must be mutually exclusive, progressive and impose bounded waiting with reference of time.

2.2 Process Definition

In this paper authors have assumed that processes are running concurrently in the distributed computer system. One can define the process, which is going to be executed in the distributed environment. The process can be classified as a macro, subprogram, subroutine or block of code etc., which has an identification number called as process_id. The UML class diagram consisting of the number of attributes and methods is shown in Fig. 2(a). The instance of the process and multiple instances through the objects are also represented as in Fig. 2(b) & 2(c), respectively. All the processes, which are compteting for getting access of shared resources, are to be arranged in a bi-directional logical ring. The instances of processes arrange in a bi-directional ring, can be shown as Fig. 2(d). This completes the definition of the process representation.

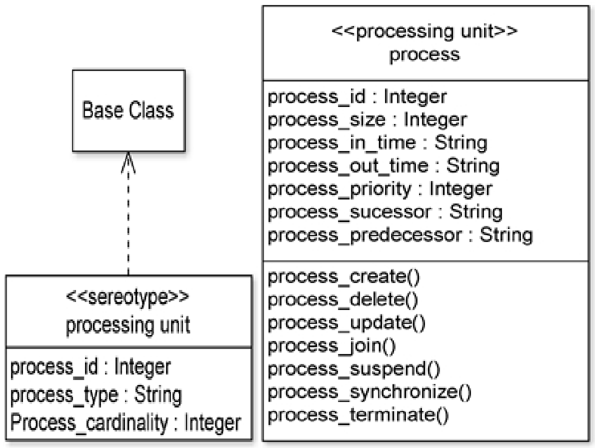

Fig. 2(a) Process UML class

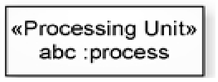

Fig. 2(b) Instance of process

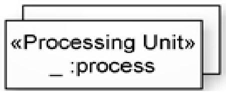

Fig. 2(c) Multiple instance of process

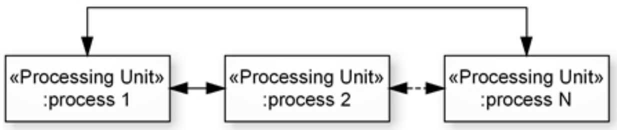

Fig. 2(d) Logical ring of process

2.3 Request_Token

The Request_Token class is a stereo typed class, LIST as a Meta class. It is shown in Fig. 3 with its attributes. The Request_Token is a list of process requests; each denotes unique process request description. Every time a new process request, get appended at the last in the Request_Token, when any process generates the request for accessing the shared resource or wants to execute its critical section.

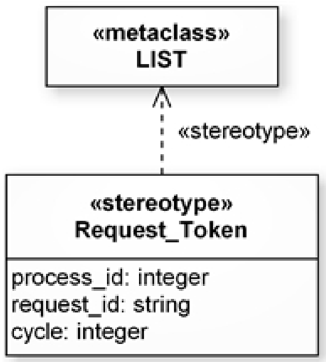

Fig. 3 Request_Token meta class

3. Algorithm for Establishing Mutual Exclusion

A number of solutions have been proposed to mutual exclusion problem for fully distributed environment but most of them imposes heavy message passing overheads on the network and based on unidirectional ring technique. The designed algorithm for establishing mutual exclusion in fully distributed environment is based on token approach using bidirectional ring technique. The assumptions and conditions, considered, while designing the algorithm are given below:

1. The distributed computer system is composed of N nodes or sites {S1, S2, S3,….SN}, equipped with its own local memory and processor.
2. The participating sites do not share any memory or global clock. The only way to get communicated is through message passing mechanism.
3. On each site, there is exactly one process is running, which can issue more than one request for accessing the critical section but only after the completion of previous one.
4. All the processes competing for accessing critical section are arranged in a bi-directional logical ring imposing in any ordering on process number i.e. Pi can exist on ring before or after Pj.
5. All the processes know their successor and predecessor in the bi-directional logical ring i.e. a process should know the network address of its next process and the previous process.
6. There is one separate bi-directional ring exists per critical section or shared resource.
7. Any new process can join the contention ring at any time by proper exchanging of their successor and predecessor addresses.
8. The joining of new process in the ring will be done by, breaking the logical link between any two processes and exchange the addresses in the following manner:
9. Suppose process Q wants to join the bi-directional ring or it is going to be insert between the existing process P and process R then,
    a. Q-> successor = P-> successor
    b. Q-> predecessor = R-> predecessor
    c. P-> successor = Q
    d. R-> predecessor = Q
10. Instead of token message, a new data structure, called Request_Token message will propagate around the bidirectional ring.
11. The Request_Token retains the following information about the process:
    a. process_id: Process identification number used to recognize the process uniquely in the distributed system.
    b. request_id: The request_id is a number that increments every time when any process generates new request for accessing the critical section.
    c. cycle: Required in case of Request_Token lost, one by default.
12. Every site maintains a token_lost flag also, FALSE by default.
13. The token will move left to right on the bi-directional ring. Only process failure recovery algorithm uses the bi-directional property of the ring.
14. On receiving of Request_Token by the process P, and then copying of the Request_Token means update the process request entries to the already existing Request_Token in the node's memory and copy it in the node's memory, if it is not found there.

The perception behind the algorithm is that, whenever any process needs to access its critical section, it has to submit its request by appending these three entries in the Request_Token. As the Request_Token follows the concept of List, so the process request entries can be made dynamically. In this way the Request_Token can be any of the length. To carry the Request_Token of arbitrary length, ring topology is an appropriate choice, because a ring can carry data of any length. On receiving of Request_Token by the process P on bi-directional ring, it will react in the following manner,

1. Process P does not want to access its critical section then it will simply copy the Request_Token to the node's memory and forward it to the next successor process.
2. Process P wants to access its critical section and it finds their request entry in the received Request_Token, then it will,
    a. Hold Request_Token
    b. Access critical section
    c. Remove the corresponding entry form Request_Token after completing its critical section
    d. Save the updated Request_Token in the node's memory.
3. Process P is in the critical section, then it will simply copy the Request_Token to the node's memory and forward it to the next successor process.
4. Process P wants to submit their request for accessing the critical section then it will,
    a. Increment the request_id of last entry in the received Request _Token, by one. If it is first process request entry then request_id =1.
    b. Append the aforesaid three entries in the Request_Token
    c. Save a copy of request-token in the node's memory.
5. Upon receiving Request_Token, any process P updates the Request_Token in node's memory, if and only if the cycle value in Request_Token >= cycle value of already existing Request_Token in the site's memory.
6. In case of Request_Token lost, a timer expires at the process. In this case the process P generates the lost_token message. This lost_token message will contain a field for saving the node address and highest

request_id value, found in the Request_Token entries stored at any node's memory. When any process Q recieves the lost_token message, it will set token_lost flag true and copy its node's address and highest request_id value found in saved Request_Token at that site, if found greater than the lost_token message already have.

7. After completing one complete cycle, the lost_token message will contain the site address, which has the largest request_id. Create a copy of Request_Token from that site; set the token_lost flag false, increment the cycle value by one, put it on the ring back.
8. When this Request_Token with incremented cycle value reaches to other process R that have lesser cycle value in saved Request_Token, replace the saved Request_Token by the newly received Request_Token and set the token_lost flag false.
9. If Request_Token arrived at a process R with the smallest cycle value than the process already have saved in its memory, then this Request_Token is assumed to be obsolete and hence discarded.
10. If lost_token message arrives repeatedly at the process R that have already token_lost flag true then these lost_token messages is assumed to be obsolete and hence discarded.
11. If the fate has been completed successfully, then certainly after N complete cycles, every process has nothing in its memory, where N is the number of competing process for accessing critical section.

3.1 Recovery algorithm from Process Failure

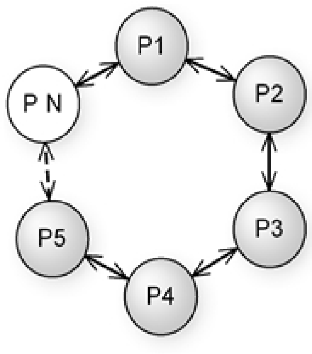

Fig. 4: Logical ring of process

In Fig. 4, one can see that processes P1, P2……PN are arranged in a bi-directional ring. Here authors have assumed that at a time only one process shall failure on the bi-directional ring, while others remain alive, in the distributed computing environment.

Suppose there are five processes on the ring and process P1 has failed then process P2 will not be able to see the Request_Token for a specified period of time. In this case it will react as follows,

1. Process P2 will send an echo packet to its predecessor process P1.
2. If it doesn't find any response from its predecessor process P1 within a certain amount of time, declare it dead.
3. Now process P2 generates a recover message, which contains two fields, sucessor_end and the predecssor_end, for storing the successor and predecessor process addresses respectively, and a list for maintaining the entry of all the live processes.
4. Process P2 sends an echo packet to the next process P3, if it gets reply from process P3, Process P2 saves its own address in successor_end, put the recover message on the bi-directional ring and passes it to the next successor process P3.
5. Upon receiving the recover message, process P3 will make its own entry in recover message and send an echo packet to its successor process.
6. If process P3 gets echo back then passes this recover message to next successor P4.
7. Repeat steps 5 and 6 for Processor P4.
8. When recover message reaches to process P5, it would not get echo back from process P1 as it is failed. In this case process P5 saves its own address in predecessor_end in recover message.
9. Replace, P5->successor address with recover message->successor_end.
10. Match the process entries in saved Request_Token with the recover message live process entries. Remove extraneous process entries, in Request_Token saved at the site's memory.
11. Change the direction of recover message and send it to the predecessor process P4. Repeat step10 and send the recover message again to the next predecessor P3.
12. Repeat step10 and send the recover message back to issuing process P2.
13. Replace the P2->predecessor with recover message->predecessor_end and Repeat step 10.

The above recovery method has been designed for one process failure at a time only. The process declared dead can join the bidirectional contention ring later at any time.

4. UML Modeling of Mutual Exclusion Algorithm

4.1 UML Class Diagram

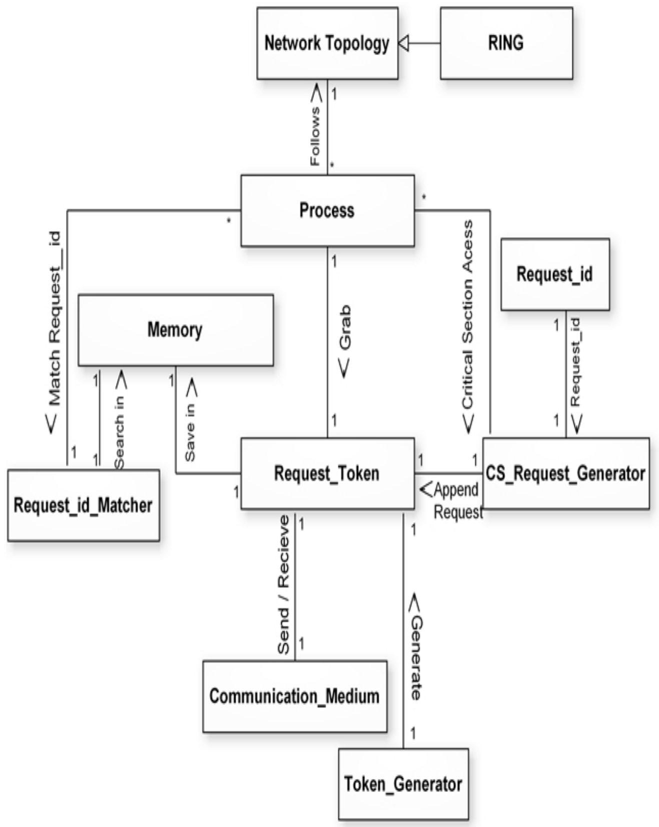

Fig. 5 UML class diagram for mutual exclusion

Fig. 5 shows the UML class diagram for mutual exclusion in distributed environment. In this diagram one can see the classes and their relation, involved in the system. Here authors have considered that the process are arrange in a bi-directional ring. The Process class is responsible for grabbing operations for the Request_Token moving around the ring. For the movement of Request_Token on the ring Reqest_Token class would interact with Communication_Medium class, which follows some sort of communication technology used for constructing the distributed system.
Also the Request_Token class will directly interact with the Memory class for performing the save and update operation for the Request_Token entries. The Memory class is responsible for all the save and search related operations for the Request_Token. Request_id_Matcher will also interact with the Memory for searching the respective request_id for the corresponding process. Token_generator class is responsible for generating new token and putting it on to the ring, in case of ring initialization and Request_Token lost.
For submitting the request for accessing the shared resource or executing its critical section, Process class will interact with the CS_Request_Generator class; which is responsible for the proper generation of request, to the corresponding process. Here authors have considered that processes are arranged in bi-directional ring, the Ring class is responsible for constructing the bi-directional logical ring of processes.

4.2 UML Sequence Diagram

4.2.1 Process Request for Accessing Critical Section

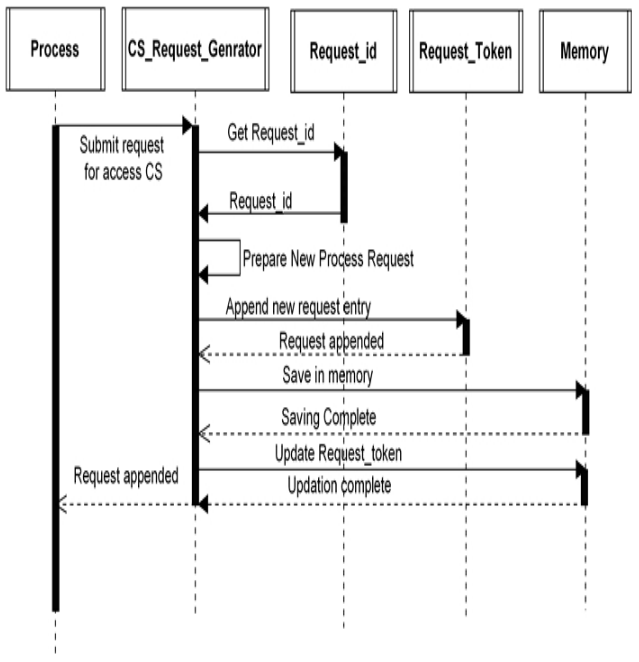

Fig. 6 UML sequence diagram for process request

In the Fig. 6, one can see that how the objects are getting activated and what are the messages being transferred among objects while submitting a process request.
If any process wants to access their critical section, a process will first submit a reqest to CS_Request_Generator object. Now CS_Request_Generator will initiate the Request_id and get the new request number. After getting new request id, CS_Request_Generator starts preparing a new process

request and append this request entry to the existing Request_Token. Now update the Request_Token in the memory and upon completion, send an acknowledgement back to the process for a successful request submission.

4.2.2 Request Granted to process for Accessing Critical Section

The Fig. 7, given below, shows the sequence of messages communicated when any process permitted to access their critical section. First of all, token is grabbed by the process and saved into the memory.
Now Request_id_matcher will start searching for respective request_id issued by the corresponding process in Request_Token, if search declared successful then process will start its critical section execution.
After the completion of critical section execution, remove the request entry of corresponding process from the Request_Token and pass it to its successor after updating the copy of Request_Token, already saved in the memory at respective site.

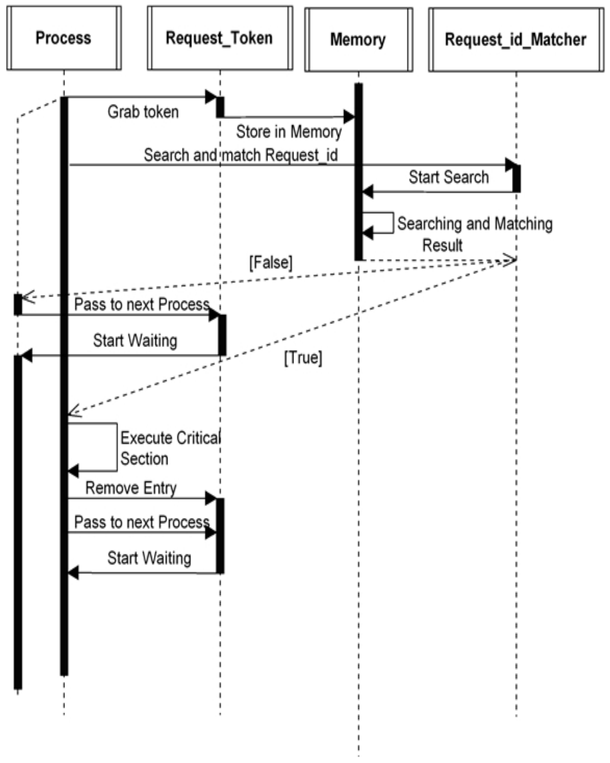

Fig. 7 UML sequence diagram for request granted to process

If search fails the process will pass the Request_Token to the next successor and start waiting again for another chance.

5. Correctness of algorithm

5.1 Mutual Coordination

This algorithm ensures the mutual coordination among the processes running concurrently in a distributed environment. Any process, which holds Request_Token, can execute their critical section only. The Request_Token will be forwarded to next successor process only when the current process will complete the execution of critical section. Therefore there is no any possibility for more than one process to enter in its critical section at a same time.

5.1 Bounded Waiting Time

The Request_Token will contain the entry of request_id for each competing process, which ensures that each process would be served according to their request_id number. As the token moves along with the ring and passes from one process to another process on the ring, every request will serve within a certain amount of time. Suppose process P is initiated a request for accessing the critical section by appending a request entry in the Request_Token. This request would be granted after completing one complete cycle of a ring i.e. when process P would receive the Request_Token again. Starvation will occur in the system, because every process will evenly get a chance to complete their request.

5.2 Lost Request_Token Recovery

The Lost Request_Token recovery algorithm given in the paper is designed to regenerate the Request_Token, in case of token lost in a distributed computing environment. When Request_Token reaches to any process, a copy of token is maintained at the respective site's memory. In case of token lost the Request_Token, copy kept at site, having the largest value of request_id, will be responsible for the new Request_Token generation. Now this new Request_Token have to be copied out to each of the participating site to overcome the token lost problem on a process ring.

5.3 Recovery from Process Failure

It is possible that during the execution, any process remain fail to pass the Request_Token to its successor process. On expiration of timer the successor process will try to find out that whether the predecessor process is alive or not. If it is found dead, it will generate the recover_message to find out the both ends, where the logical link has been broken. Once it finds out both the ends of ring, one can join the broken link by exchanging the successor and predecessor addresses. Removing the entry of the dead process(s) from the Request_Token will prevent serving of request issued from the dead process(s). The above process failure recovery algorithm has been designed for only one process failure at any instance on the bi-directional ring.

5.4 Ensure Progress

As this algorithm, provides recovery mechanisms from both of the hurdle i.e. token failure and process failure, of token-based approach for establishing the mutual exclusion in distributed environment. This will ensure the progress of process execution results at higher rate of concurrency in any distributed computing scenario.

6. Performance Analysis

6.1 Message Complexity

If there are N number of processes are running concurrently in the distributed computer system, then one can compute the message complexitys as shown in Table 1.

Table 1: Message Complexity

Activity	***Messages***
Lamport's Algorithm [3]	3(N-1)
Ricart-Agarwals Algorithm [4]	2(N-1)
Maekawas Algorithm [5]	3√N
Token Ring Algorithm [6]	Avg N/2
Suzuki-Kasamis Algorithm [7]	N
Raymonds Algorithm [8]	Avg 2LogN
Fahims Algorithm [9]	N
Author's Algorithm	N

As the present approach is a token based approach uses bi-directional ring technique, in comparison of all previous approaches are based on unidirectional ring technology. To deal the problem of establishing the mutual exclusion in the distributed computing environment, the older approaches have adopted the request message collection methodology, which imposes heavy message overheads on communication network. Besides that in the present approach there will be less communication overheads on the network due to Request_Token approach on the bi-directional ring

7. Concluding Remarks

In this paper, authors have designed, a new protocol based on Request_Token approach using bi-directional ring for ensuring the mutual exclusion in a fully distributed computer system through Unified Modeling Language (UML). This design is equipped with the mechanisms to recover the system from token lost and process failure. Authors have also given the correctness and message complexity for this algorithm, and found satisfactory. In this way one can say that UML plays an important role in visual representation of any software and hardware problems. Bottom up approach used in UML modeling makes it more robust, reliable and efficient.

References

[1] Siberschatz, A., Galvin, P. B., 2000, Operating Systems Concepts, 5th edition, John Wiley & Sons, Inc.

[2] Silberschatz, A., Peterson, J.L., 1988, Operating System Concepts, Addison-Wesley, Alternate edition.

[3] Lamport, L., 1978, Time, clocks, and the ordering of events in a distributed system, Communications of the ACM, vol. 21, no. 7, July 1978, pp. 558-565.

[4] Ricart, G., Agrawala, A., 1981, An optimal algorithm for mutual exclusion in computer networks, Communications of the ACM, vol. 24, no. 1, Jan. 1981, pp. 9-17.

[5] Maekawa, M., 1985, A sqrt(n) algorithm for mutual exclusion in decentralized systems," ACM Transactions on Computer Systems, vol. 3, no. 2, May 1985, pp. 145-159.

[6] Agrawal, D., El Abbadi, A., 1991, An efficient and fault-tolerant solution for distributed mutual exclusion, ACM Transactions on Computer Systems, vol. 9, no. 1, Feb. 1991, pp. 1-20.

[7] Suzuki, I., Kasami, T., 1985, A distributed mutual exclusion algorithm, ACM Transactions on Computer Systems, vol. 3, no. 4, Nov. 1985, pp. 344-349.

[8] Raymond, K., 1989, A tree-based algorithm for distributed Mutual Exclusion, ACM Transactions on Computer Systems, vol. 7, no. 1, Feb. 1989, pp. 61-77.

[9] Kawsar, F., Shaikot, S.H., Saikat, S., Mottalib, M.A., 2002, An Efficient Token Based Algorithm for Mutual Exclusion in Distributed System, Proceedings of the 5th International Conference on Computer and Information Technology

(ICCIT 2002), pp. 93-96, Dhaka, Bangladesh, December 2002.
[10] Chakraborty, R.N., Yaprak, E., 1993, Improvement in reliability of the token ring network by reversal of token in case of a single component failure, Circuits and Systems, IEEE, Proceedings of the 36th Midwest Symposium on Vol. 2 , Issue, 16-18 Aug 1993 pp. 1152 - 1154.
[11] Andrew S. Tanenbaum, 1995, Distributed Operating Systems, Prentice Hall, 1995.
[12] Booch, G., Rumbaugh, J., Jacobson, I., 1999, The Unified Modeling Language User Guide, Addison Wesley, Reading, MA 1999.
[13] OMG, 2001, Unified Modeling Language Specification. Available online via http://www.omg.org.
[14] OMG, 2002, OMG XML Metadata Interchange (XMI) Specification. Available online via http://www.omg.org.
[15] Pllana, S., T. Fahringer, 2002, On Customizing the UML for Modeling Performance Oriented Applications. In <<UML>>, Model Engineering Concepts and Tools, Springer-Verlag., Dresden, Germany.
[16] Pllana, S., T. Fahringer, 2002, UML Based Modeling of Performance Oriented Parallel and Distributed Applications, Winter Simulation Conference.
[17] Saxena, V., Arora, D., Ahmad S., 2007, Object Oriented Distributed Architecture System through UML, conference IEEE, International Conference on Advances in Computer Vision and Information Technology, ACVIT-07, ISBN 978-81-89866-74-7, pp.305-310.

Dr. Vipin Saxena: He is a Reader & Head, Dept. of Computer Science, Babasaheb Bhimrao Ambedkar University, Lucknow, India. He got his M.Phil. Degree in Computer Application in 1991 & Ph.D. Degree work on Scientific Computing from University of Roorkee (renamed as Indian Institute of Technology, India) in 1997. He has more than 12 years teaching experience and 16 years research experience in the field of Scientific Computing & Software Engineering. Currently he is proposing software designs by the use of Unified Modeling Language for the various research problems related to the Software Domains & Advanced Computer Architecture. He has published more than 55 International and National publications.
Phone: +91-9452372550
Fax: +91-522-2440821
E-mail: vsax1@rediffmail.com

Deepak Arora: He is a Research Scholar, Dept. of Computer Science, Babasaheb Bhimrao Ambedkar University, Lucknow, India. He got his Master Degree of Computer Applications in 2003 and M.Phil. Degree in Computer Science in 2006. Currently he is actively engaged in the research work on Distributed Computing Systems through the Unified Modeling Language. Ha has produced several outstanding publications on Distributed Computing Systems.
Phone: +91-522-2439567, +91-9889505518
E-mail: deepakarorainbox@gmail.com