System Design Document for the Lab3 - Eventual Consistency SDD

replica A	**replica B**	**replica C**
hi <1,A>	hi <1,A>	hi <1,A>
how are you <2, A>	How do you do <2, B>	How are you guys <2, C>
How do you do <2, B>	how are you <2, A>	How do you do <2, B>
How are you guys <2, C>	How are you guys <2, C>	how are you <2, A>

after a get request, the timestamp of the node with the highest id breaks the tie.

replica A	**replica B**	**replica C**
hi <1,A>	hi <1,A>	hi <1,A>
how are you <2, A>	how are you <2, A>	how are you <2, A>
How do you do <2, B>	How do you do <2, B>	How do you do <2, B>
How are you guys <2, C>	How are you guys <2, C>	How are you guys <2, C>

so what we do is we keep an Events log-list and a timestamp variable locally in each replica.

- each time a replica does an operation, it creates an object of Event (operation type A/M/D, timestamp, sender-node-id, Msg <id,msg>) ex. (A, 3, 5, <2, "How are you?">), and propagate the same information of the event to all the replicas.
- whenever a replica gets an event that has a timestamp that is higher than its timestamp it does, operate, create event, add it and update the local timestamp to the message timestamp.
- whenever a replica gets an event that has a timestamp that is lower than its timestamp it does the following:

    - it goes to its list of log-events, apply the undo-function on all the events in the list from the end of the list back, it stops undoing whenever it reaches a timestamp lower than or equal to the incoming timestamp.
    - When it has finished undoing, it compare the timestamp with the first undone operation,
        - if the timestamp of the message is higher than the one in the list, execute the message operation and att the event to the log.
        - if they are same it compare by the node id with while loop ging back to the first event with node_id lower than the incomings message node id, it then does the operation and add the event
        - then, in both cases it redo all the next events in the list.
- The undo-function:
    - If it the event to undo is add, we simply delete the entry with this id from the board
    - If the event is modify, we go back in the list to find the last done operation to the same message id,
        - if it was modify we do it,
        - if it was add we simply
            - delete the modified version (the entry at all) and
            - add it again with the same id (apply the operation).
    - If the event is delete,
        - we go back in the list to find the last done operation to the same message id,
            - if it was modify we add a new entry with the same information in the modified event,
            - but if it was add we simply apply the add event.
- For object orientation we create apply-event-function that takes an event object and applies it locally in the replica.

- when two networks merge by a node then:
    - a merge message sends to the merging node with all the log of the network2 to the merging node, and with the list of nodes of the network2. The merging node does 2 things:
        - appand the network2 node list to its list and sends to all the nodes the new nodelist to update msg.
        - loop over its own log-list and propagate the events event after the other, to the list of nodes of network2.
        - loop over the incoming log list of network2 and propagate it to the all the nodes in the network1 including it self.

The complexity of our design:

- we consider that we have **n** nodes and in each node we have **m** operations in the log list.
- the complexity as it described in the state-diagram,
    - wc=bc=n-1 + 4 = O(n)

      +
    - wc=2m-1 * (n-1) = O(m*n)

====> O(m*n)