Palak Bhasin, Nicholas Hyland
CSCI-GA.2434-001 – Fall 2017 – Advanced Database Systems
Replicated Concurrency Control and Recovery – Design Document

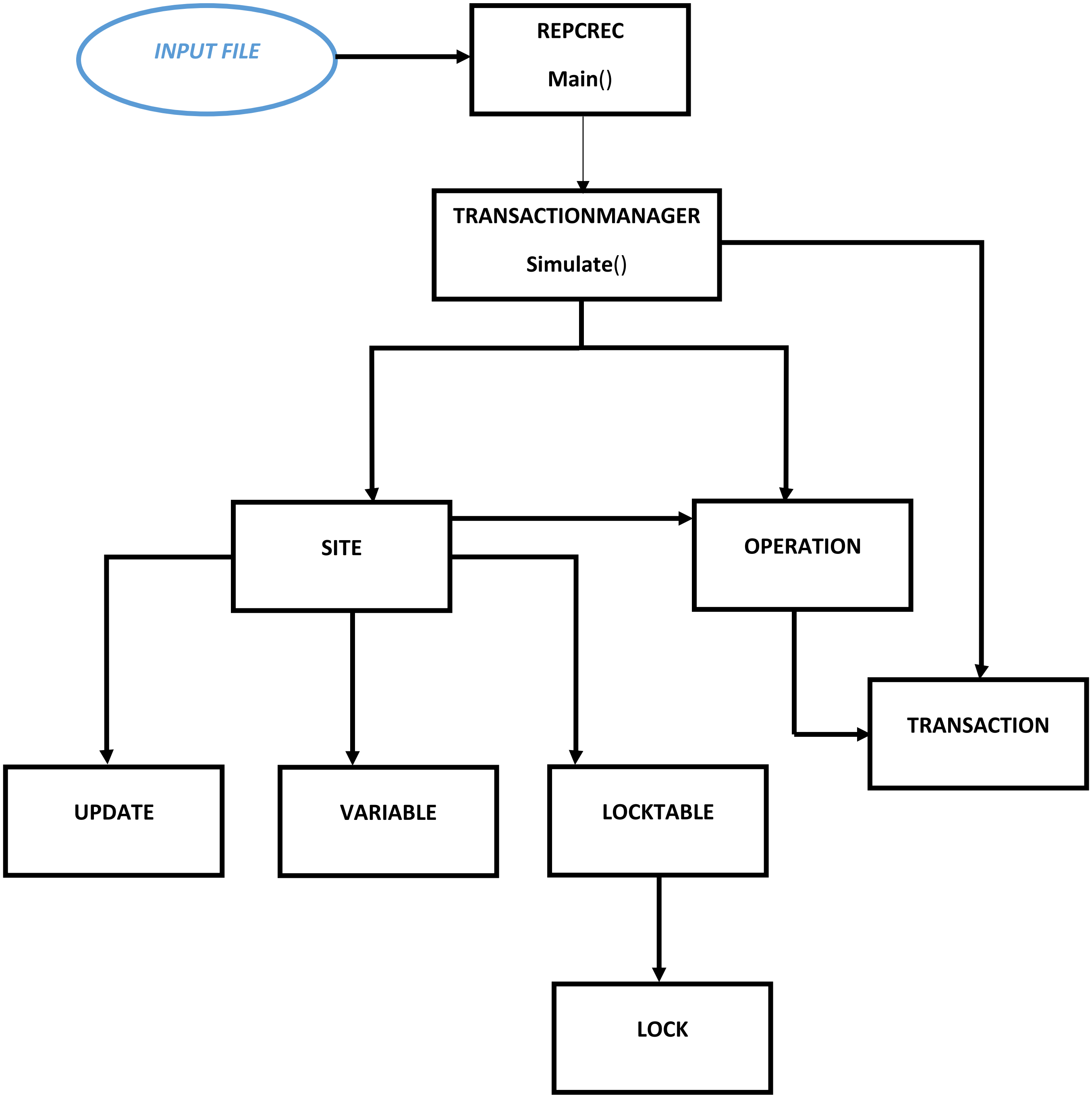
REPCREC is a java project that models a distributed database and implements the available copies approach, strict two-phase locking, multiversion concurrency control, deadlock detection, replication and failure recovery.

Objects:

- Lock
  - Stores the ID of the transaction holding this lock, the lock start time, and whether it's a read or write lock.
- LockTable
  - Each lock table is local to a site and consists of three hashmaps (with variable IDs as keys):
    - writeLocks: each variable maps to a single transaction holding a lock
    - readLocks: each variable maps to a list of transactions holding locks
    - lockQueue: each variable maps to a list of pending locks
- Operation
  - Each line that is read in from the test file is created as an operation
  - Types: begin, beginRO, R, W, end, fail, recover, dump
- Site
  - Each site has a list of variables, an independent lockTable, and a list of updates (to keep track of commit history at this site)
  - The list of variables is initialized based on the site's number (ID: 1-10)
    - If the number is even, all even variables and 2 odd variables (number-1, number+9) are added to the site's array list of variables.
    - If the number is odd, all even variables are added to the site's array list of variables.
  - Keeps track of whether the site is currently down or has failed in the past and the time it most recently failed/recovered
  - At failure
    - the lock table is erased
  - At recovery
    - all values committed before failure time are preserved
    - non-replicated data is available for reads and writes immediately
    - replicated data is available for writes immediately, not available for reads until a write has been committed
  - getLatestValueRO(Operation o, int Time)
    - returns the latest committed value from when this operation began
- Transaction
  - An operation that reads or writes
  - Each transaction consists of a list of pending (uncommitted) operations
- TransactionManager
  - Processes each operation passed in as a list from RepCRec
  - Keeps track of 3 lists: operations, blockedOperations and sites

- Keeps track of 2 hashmaps: transactions and graph (of conflicts)
- Initializes a list of 10 sites
- simulate():
    - A while loop iterates until the lists of operations and blockedOperations are both empty
    - During each loop, time increments by 1 and an operation is either processed completely, or added to the list of blockedOperations if the read or write cannot be processed at the current time
    - At each tick, all operations in blockedOperations are processed first
    - Calls the appropriate function to process each operation based on Operation type:
        - beginTransaction(Operation o, Boolean isReadOnly)
            - adds a new Transaction object to the transactions hashmap
        - endTransaction(Operation o)
            - If the transaction is read only, do nothing.
            - Else, abort the transaction if necessary
            - Else, commit all values from the pending operations for this transaction
            - Drops all locks and clears all conflicts for this transaction
        - write(Operation o)
            - Checks if sites are down or locked depending on the variable
            - If it can successfully retrieve the locks for this transaction, calls addGraphConflicts and return true
            - Else, adds the transaction to the lock queue and returns false
        - read(Operation o)
            - If the transaction is read only, process it if possible
            - Checks if sites are down or locked depending on the variable
            - If it can successfully retrieve the locks for this transaction, calls addGraphConflicts and return true
            - Else, adds the transaction to the lock queue and returns false
        - dump(Operation o)
            - Output the state depending on whether a dumpVariable or dumpSite was provided
        - failSite(Site s)
            - Calls site.fail(currentTime)
        - recoverSite(Site s)
            - Calls site.recover(currentTime)
    - After a read or write operation is processed, calls detectDeadlock():
        - Calls detectCycleStart(graph):
            - It returns the cycle, or an empty list depending on whether or not there is a cycle in the graph
                - If there is no cycle, then there is no deadlock and it returns

            - If there is a cycle, it will look at all the transactions, given by their ID in the arraylist, and determine which transaction is the youngest
        - Once the youngest transaction is detected, it is aborted and it's locks and conflicts are cleared
- Update
    - Stored at each site per variable
    - Stores the value and time for each commit
- Variable
    - Has an ID, value and latest commitTime


INPUT FILE
REPCREC
Main()
TRANSACTIONMANAGER
Simulate()
SITE
OPERATION
TRANSACTION
UPDATE
VARIABLE
LOCKTABLE
LOCK