Design of RADEx - Real-time Active Database Experimental System [1]

Rajendran M. Sivasankaran & Bhaskar Purimetla
Directed by:
John A. Stankovic, Krithi Ramamritham & Don Towsley
Department of Computer Science
University of Massachusetts, Amherst, MA 01003

1 Introduction

This report presents the design of an active real-time database simulator. In section 2 we discuss the simulator architecture, a high level overview of modules and their functionality. In section 3 we discuss the individual modules in a detailed manner by describing the various parameters that control their behavior as well as their interfaces with other modules. Details of the design of every module is accompanied by a brief description of the current status of the module. Appendix I consists of the DeNet [1] declaration of the parameters along with sample values. Some parameters may not have any description because of their relative insignificance. Appendix II gives the list of files that constitute the simulator and a brief description of the files. The simulator is under continuous development with current extensions in the areas of logging, recovery and buffer management.

2 Architecture

Let us briefly discuss the modeling of the passive data in the simulator. The database is modeled as a collection of multi-level objects. The number of levels is restricted to two for ease and simplicity. So the database contains a collection of objects belonging to different classes. Each object has a certain number of predefined methods. The object can be accessed only through the methods. The methods of the parent object may invoke the methods of the subobjects which can be either read or write. This multi-level nature of the objects and the operations will provide the necessary features in the simulator to experiment with multi-level concurrency control algorithms. The high level design of the simulator is illustrated in Figure 1. The figure also presents the communication/control structure of the simulator. The simulator consists of six major components. The components along with the major functions they simulate are as follows:

- *Source*:

  The source (aka transaction generator) simulates the incoming transactions into the system. It actually simulates the application or the environment in which the Real-Time Database is used. The transaction generator can be changed depending on the application we are modeling. We can study the transaction characteristics of the application of our

[1]This work was supported in part by NSF under grants IRI-9114197 and IRI-9208920.

choice and design the generator to model it as closely as possible. It generates transactions with timing constraints with a particular arrival distribution. It can generate both periodic and aperiodic transaction streams. We refer to the transactions generated in the source as external transactions.

- *Transaction manager*:

  Transaction manager receives the transactions from the source generator. It executes the submitted transactions by calling object manager to execute the methods of various objects accessed. It handles the various transaction events like begin, commit and abort. It also does load balancing and transaction priority assignment. Various priority assignment algorithms can easily be plugged in and evaluated. It also informs the rule manager of the transaction events.

Components of the simulator

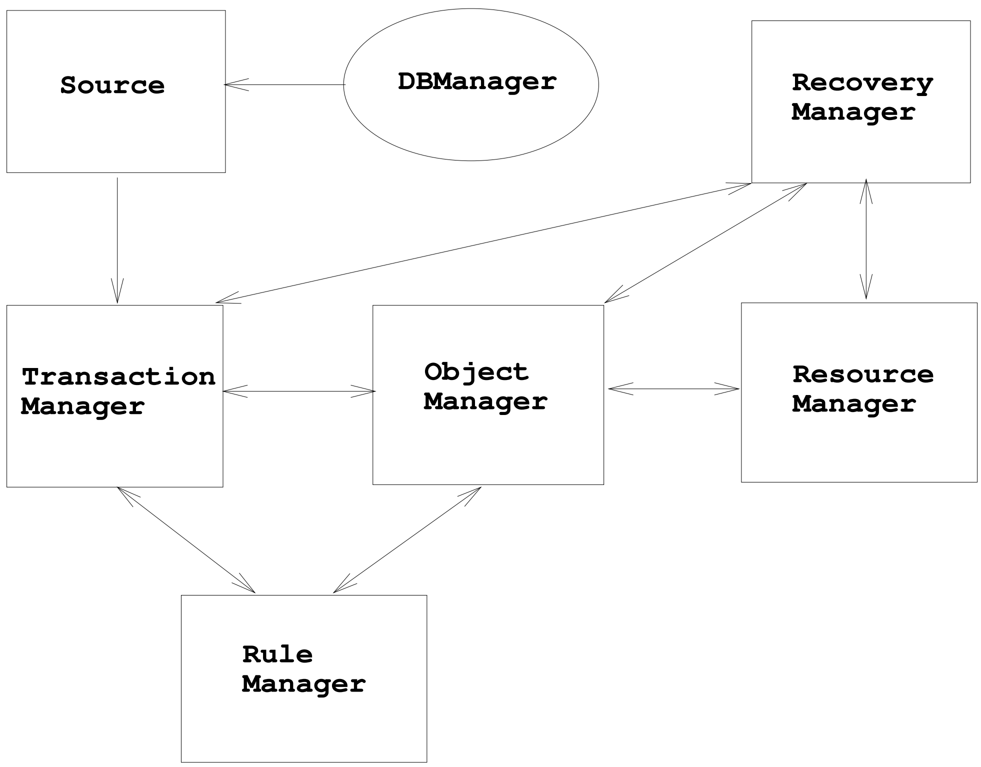

Figure 1

- *Object Manager*:

  Object manager executes the methods of the objects in the system as requested by the transactions. It is also responsible for concurrency control. If the request can be executed it executes and sends the success message back to the entity executing the transaction. If the transaction request cannot be satisfied it either sends the blocked or abort message back to the executing entity. It also informs the rule manager of the various basic object events taking place.

- *Rule Manager*:

  Rule manager checks to see if any rules are triggered whenever it gets an event notice from the transaction manager or the object manager. It also models condition evaluation. It generates the transactions corresponding to the actions of the rules triggered if their conditions are satisfied and submits them to the transaction manager. The rule manager models the active workload in the system.

- *Resource Manager*:

  The resource manager simulates the CPUs and Disks. It also simulates the main memory buffer. The Object manager requests the resource manager for the necessary pages or for CPU time. If the particular page is not in the memory buffer the resource manager brings it into memory. Different buffer replacement algorithms can be plugged in into the resource manager. Similarly, we can experiment with various disk scheduling algorithms.

- *Recovery Manager*:

  The recovery manager module is responsible for rollbacks and recovery of transactions. The transaction manager informs the recovery manager of the transaction events. The object manager informs the recovery manager of all the high level object operations and low level page operations. The recovery manager is responsible for logging the updates at page level and compensation information for the high level operations. This preserves the ACID properties of the multilevel transactions.

3 Module Parameters and Detailed Description:

In this section we will discuss the various components of the simulator in a detailed manner. We will also discuss the various parameters which control the behavior of the modules. First let us discuss the data model used and then the components.

3.1 Data Model

The module that models the data is called the DBmanager. The data is modeled as having certain number of object classes and each object class having certain number of instances. We model the objects as multilevel objects, but restrict the number of levels to two. Each object class has certain number of methods defined which are used to access the object. Each object instance in the database is mapped to a certain number of pages in the secondary storage. The first page for an object instance is the header page and is assumed read-only to avoid hotspot accesses. The subobjects are assumed to be of equal size and stored in sequence. The number of pages that each object instance in a class needs is given as a parameter. All object instances of a particular class are assumed to be stored in a single file. So, now we have a collection of files of known size. We have to build block directories for each file. This is done at initialization time. We can select all the pages which form a file randomly. We can change this to have a sequential or any other organization, with which we would like to experiment.

3.1.1 Current Status

Currently the data model is just made up of a number of object classes having certain number of instances. The multi-level implementation of the objects is not incorporated in the current version of the simulator. Note that the *subobject* parameters are not yet used in the simulator.

3.1.2 DBManager Parameters

- *MAXOBJCLASSES*: This is the maximum number of object classes that is in the database.
- *numberOfClasses*: This parameter defines the actual number of object classes to which the objects in the database belong.
- *numInstances*: This is a per class parameter that gives the number of object instances in each class in the database. The sum of all class instances gives the total size of the database.
- *numSubObjects*: This per class parameter controls how many subobjects belong to an object instance belonging to that class.
- *numMethods*: This per class parameter controls the number of methods an object instance of that class will have.
- *numPages*: This per class parameter gives how many data pages an object of that particular class span.
- *dataContention*: This per class parameter helps us tune the data contention for each class.
- *methodProbability*: This per class per method parameter gives the probability of a particular method for a class.
- *methodCPUTime*: This per class per method parameter gives the execution time of a particular method for a class.
- *numSubObjAccessed*: This per class per method parameter gives the number of subobjects accessed by a method for a class.
- *updateProb*: This per class per method parameter gives the probability that a particular method of a class causes an update to that object.
- *replicaSite*: This per class per site parameter tells if a particular class of objects is present in a given site.

3.2 Source

The transaction generator generates the transactions and sends them to the transaction manager for execution. A transaction generator can either generate a stream of aperiodic transactions based on a Poisson arrival pattern or any other distribution pattern. It can also generate a periodic stream of transactions. So in an instance of a simulator there will be a number of instances of transaction generator module simulating different streams of transactions. The transaction generator also can generate either hard real-time or soft real-time transactions. The transaction generator also generates the external events signaled from the applications as part of the transactions. Here we assume that we are generating transactions at a lower level after the queries have been processed using indexes and all the object instances which are to be used have been identified. If we want to model index usage then we can have a higher level query and probabilistically select the objects we are going to access and model index usage also probabilistically by assuming a B+ tree structure.

3.2.1 Source Parameters

- *MAXTRANSCLASS*: This specifies the maximum number of transaction classes in the system.
- *InPeriodic*: This parameter controls whether this instance of the source outputs a stream of aperiodic transactions or a periodic stream of a single transaction type.
- *interArrival*: This specifies the interarrival time of aperiodic transactions.
- *InHardFirmsoft*: This parameter specifies whether this stream generates hard real-time, firm real-time or soft real-time transactions.
- *closed*: This specifies if the stream is closed queue or open queue model.
- *classExtEvents*: This per class parameter gives the number of external events triggered by that class of transactions.
- *classPriority*: This per class parameter gives the priority of that class of transactions.
- *arrivalDistribution*: This parameter gives what arrival distribution the source follows in generating transactions.
- *deadLineRatio*: A multiplier used to control the deadline value.
- *numOfTerminals*: The number of terminals at the source.
- *maxTransPending*: The maximum number of unfinished transactions that can be submitted from the source to the Transaction Manager.
- *thinkMean*: The mean time between the generation of two transactions.
- *numTransClasses*: This parameter is equal to 1 if this is a periodic stream. Otherwise this parameter specifies the number of different classes of transactions this module generates.

- *dedicated*: Not Used.
- *minSlack*: A per class parameter that gives the minimum slack values of that transaction class.
- *maxSlack*: A per class parameter that gives the maximum slack values of that transaction class.
- *classProb*: A per class parameter that gives the probability of generation of that class of transactions.
- *maxClassNObjs*: This per class parameter specifies the maximum number of objects a transaction of that class touches.
- *minClassNObjs*: This per class parameter specifies the minimum number of objects a transaction of that class touches.
- *classTouchAll*: This per class parameter specifies if the transaction of that class touches all the objects.
- *classCpuPerRec*: Not Used.
- *classSequential*; This per class parameter specifies if the transactions of that class are sequential.
- *classAccess*: This per transaction class, per object class parameter gives the probability of access for a given object by a given transaction of a particular class.
- *classMeanSize*: This per transaction class, per object class parameter gives the mean number of objects of that class touched by a transaction of that class.
- *sequentialAcc*: Not Used.
- *classWriteProb*: This per transaction class, per object class parameter gives the probability that an object of that class is updated by a transaction of the specified class.

3.3 Rule Manager

The rule manager is one of the complex modules in the system with a lot of parameters. The rule manager models the rules probabilistically. It is informed of the basic transaction and object events by the transaction manager and the object manager, respectively. The rule manager finds whether the given event triggers any rules in the database. If no rules are triggered it returns a message to proceed to the transaction and object managers. If any rules are triggered the rule manager finds whether the conditions associated with the rules are satisfied. If the condition is satisfied the corresponding action submitted as a transaction to the transaction manager. Depending upon the coupling modes the transaction manager is instructed to execute the action transaction appropriately. The coupling mode between the event and the condition as well as between the condition and the action are selected probabilistically. Three types of coupling: immediate, deferred and independent are allowed. The coupling modes affect the condition checking and action execution as follows:

- *Immediate:* If the coupling between event and the condition portions of a rule is immediate then the condition checking is done as a subtransaction of the parent transaction which triggered the rule and is executed immediately. The parent is suspended while the subtransaction is being executed. If multiple rules are triggered and all of them have immediate coupling then they are executed concurrently. If the coupling mode between condition and action is immediate then the action is executed as a subtransaction immediately following the completion of condition evaluation. The parent transaction is suspended during the execution of the immediate subtransaction.

- *Deferred:* In this coupling mode the condition evaluation subtransaction is evaluated just prior to the commit of the parent transaction. Similarly the subtransaction containing the action is executed just before the commit of the parent transaction if the Event-Cond coupling was immediate and immediately after the condition evaluation if the E-C coupling was either deferred or independent.

- *Independent:* As the word implies the condition or the action is executed as an independent transaction from the parent transaction or the condition evaluation, respectively. Here we can again distinguish two cases depending upon whether the independent transaction develops a commit dependency on the previous transaction or not. If the independent transaction develops a commit dependency then it can commit only if the original transaction which causes this commits.

The rule manager also has to take care of the cases where multiple rules are triggered by a single event and the case where a rule evaluation triggers another rule (cascade). To prevent infinite chains we put a bound on the maximum length of the triggered chain. Also the rule manager has to take care of events based on time which can be periodic or aperiodic. The rule manager also has to take care of the case where the events are not simple but complex. For example composite events can be built by the conjunction or the disjunction of simple events. Firing of composite rules can be taken care in a probabilistic fashion. Every rule fired can have an action or can be part of a composite rule in which case there won't be an action. This probability should be enough to take care of composite rules. If necessary the firing of composite rules can be made more detailed.

The rule manager is also responsible for the feedback transactions that will be fired by the environment once certain events are actuated. Our system is a closed loop system and there are going to be transactions that are generated by the feedback mechanism. Some of the rules in the rule manager are these kind of feedback rules and they will request the TM to generate some transactions that closely resemble such feedback transactions.

3.3.1 Rule Manager Parameters

- *MAXTRANSCLASSES*: Maximum number of source transaction classes.

- *MAXCONDTRANSCLASSES*: Maximum number of condition checker transaction classes.

- *MAXACTIONTRANSCLASSES*: Maximum number of action transaction classes.

- *MAXRMTTRANSCLASSES*: Maximum number of rule manager transaction classes (MAXCONDTRANSCLASSES + MAXACTIONTRANSCLASSES).

- *classTransEventProb*: This per source class parameter has 3 values for each transaction class. The three values specify the probability that the occurrence of Start, Commit or Abort of a transaction belonging to that class triggers a rule in the system. This can be used to model certain classes which are very crucial and hence trigger a lot of rules.

- *classObjectEventProb*: This per class parameter specifies for each object class what is the probability that a method invocation on that object triggers a rule in the system.

- *classActionProb*: This per class parameter specifies what is the probability that an action is invoked after a condition is evaluated.

- *classECCouplingModeProb*: Specifies the probability of different coupling modes such as IMMEDIATE, INDEPENDENT and DEFERRED for event-condition (EC).

- *classCACouplingModeProb*: Specifies the probability of different coupling modes such as IMMEDIATE, INDEPENDENT and DEFERRED for condition-action (CA).

- *CascadeEventProb*: This per class parameter gives the value of probability that the execution of an action triggers another rule.

- *classMaxChain*: This per class parameter gives the length of the maximum chain for that class.

- *classMaxRules*: This per class parameter gives the maximum number of rules that can be fired by a rule manager transaction of that class.

- *classMultipleRuleProb*: This parameter gives the probability that a rule triggering actually triggers multiple rules. That means once it is decided using the three event probabilities mentioned above that the event triggers a rule then with this probability it triggers multiple rules.

- *classAvgMultipleRules*: When a multiple rule condition is true on the average this many rules are fired on the occurence of a single event.

- *tempCouplingModeProb*: Coupling mode probabilities temporal rules.

- *ShareProb*: This controls the probability that the same object is accessed by two adjacent transactions.

- *RestartRepeatProb*: Probability that all transactions of an aborted job are repeated on restart.

- *NumAperiodicRules*: Number of aperiodic rules for temporal events.

- *NumPeriodicRules*: Number of periodic rules for temporal events.

- *MinPeriod*: Minimum period for periodic rules.

- *MaxPeriod*: Maximum period for periodic rules.
- *MeanTimeEvent*: Mean time between activation of aperiodic rules.
- *NumCheckerXactClasses*; The actual number of checker transaction classes. The Checker transactions are read-only.
- *NumActionXactClasses*: The number of action transaction classes.
- *HardFirmSoft*: The per rule manager transaction class parameter that specifies the kind of transaction that class generates.
- *DeadLineRatio*: The per class deadline ratio.
- *CheckerClassProb*: The probability of generation of a checker transaction of that class.
- *ActionClassProb*:The probability of generation of an action transaction of that class.
- *ClassPriority*: The per rule manager transaction class parameter that specifies the priority of a transaction of particular class.
- *ClassTouchAll*: The per class parameter that specifies if the transaction of that class touches all the objects.
- *MaxObjCount*: The per class parameter that specifies the maximum number of objects touched by a transaction of that class.
- *MinObjCount*: The per class parameter that specifies the minimum number of objects touched by a transaction of that class.
- *ObjClassAccessProb*: The per rm transaction class per object class parameter that specifies probability of access of a particular object class by a particular transaction class.
- *ClassWriteProb*: The per rm transaction class per object class parameter that specifies probability of update of a particular object class by a particular transaction class.
- *minSlack*: The per class minimum slack value.
- *maxSlack*: The per class maximum slack value.
- *dLinePolicy*: The deadline policy used to schedule transactions.

The parameters that control the generation of active load by the rule manager are very similar to those parameters which control the transaction generator. The rule manager can also generate periodic transaction load. To model temporal events in the rule manager we have some additional parameters. For the aperiodic events the next time instant the event is triggered is selected randomly from a uniform distribution around a mean value given by *Meantimeevent.*

3.4 Transaction manager

The Transaction Manager (TM) is responsible for assigning priorities to the incoming transactions. The TM on accepting a transaction from any of the sources assigns it to an entity (in DeNet it is a Device Module) that is capable of executing it. There will be multiple transactions (entities) that are executing in our system at any given time. A typical transaction in our system is a sequence of methods on objects. The executing entities send the request to execute the method on an object to the Object Manager (OM). If the method can be executed then the OM executes and sends a success or failure message to the responsible entity. The TM schedules transactions depending on their priority. The TM is also responsible for load control in the system. It limits the number of transactions it actively considers to schedule at any time to a maximum value. The TM is responsible for the BEGIN, COMMIT and ABORT of the transactions. It sends a message to the Rule Manager (RM) when a transaction event occurs, i.e., when a BEGIN, COMMIT or ABORT of a transaction happens.

3.4.1 Transaction Manager Parameters

- *rejectionPolicy*: This specifies the rejection policy in the transaction manager (e.g. all eligible, not tardy, feasible deadline).
- *masterLoadIo*: The I/O needed in terms of disk pages accessed to load the master.
- *masterLoadCPU*: The cpu time needed to load the master.
- *cohortLoadIo*: The I/O needed yto load the cohort.
- *cohortLoadCPU*: The cpu time needed to load the cohort.
- *maxMaster*: The maximum number of master that can be active in the transaction manager.
- *delayRestart*: This parameter specifies if there will be delay in the restarting of transactions.
- *logBlockSize*: The log block size.
- *logFlushTime*: Time to flush the log.
- *logTimeOut*: Time inbetween logflushes.

3.5 Object Manager

The Object Manager (OM) is responsible for accepting the access requests from the Transaction Manager, concurrency control and satisfying the access requests. The access requests just specify the object that is to be accessed and the method to be executed on the object. The OM checks if the object is accessible and if it is accessible, executes the method and sends back a success or failure message to the Transaction Manager (TM). If the object is not accessible it sends back a suspend message to the TM. If the object is accessible it requests the resource manager for the object. The TM also requests OM's permission to commit or abort. The object manager maps the objects to the pages and requests the resource manager for the needed pages. Mapping is done using the file directories created at initialization time. If an object has m subobjects and occupies n pages then the ith subobject will be in the page number $\lceil \frac{n-1}{m} \cdot i \rceil$.

3.5.1 Current Status

In the current version of the simulator the i/o time needed for object access is calculated from the number of pages an object spans. A more detailed buffer manager will be implemented along with the multilevel objects.

3.5.2 Object Manager Parameters

- *ccCpu*: cpu time taken to make a CC request.

Other parameters will depend on the concurrency control algorithm used in the OM. The current CC algorithm aborts the lower priority transaction.

3.6 Resource Manager

The Resource Manager (Res. M) models the physical resources of the RTDB. It includes the CPU, the disks, the buffer pool and the main memory. One disk is used for sequential logging. The recovery manager requests the resource manager to do logging at commit points of transactions. The resource manager also satisfies the object manager's requests for pages from secondary storage. The desired page is brought into the main memory buffer if already not in the buffer. We can use different buffer management strategies and various sophisticated real-time disk scheduling algorithms.

3.6.1 Resource Manager Parameters

- *cpurate*: Number of units of cpu time per unit of simulation time.
- *numberOfCpus*: Number of CPUs in the system.
- *numberOfDisks*: Number of disks in the system.
- *diskTransferRate*: Transfer rate of the disks.
- *minSeekTime*: Minimum disk seek time.
- *maxSeekTime*: Maximum disk seek time.
- *sendReceiveTime*: Time to send or receive message to another site.
- *localMesTime*: Time to send or receive message to the same site.
- *hitRatio*: the probability of finding a page in the buffer.

3.7 Recovery Manager

The transaction manager and object manager supply the recovery manager with enough information about the transaction so that the recovery manager can log all the changes and be able to provide transaction atomicity. So whenever a transaction is started the TM informs the recovery manager. The object manager informs the recovery manager of the methods being executed and the names of the compensation methods for logical undoing if the transaction is aborted. Since the objects are multilevel we need different types of logs at different levels. Logical logs at higher object level and physical redo logging at the subobject level. When a transaction is to be committed the TM calls the recovery manager, which in turn flushes all the logs to the secondary storage appropriately and then informs the TM to commit. When a transaction is to aborted the TM invokes the recovery manager, which undoes the transaction. The time taken to undo is modeled through the use of compensation operations which take same amount of time as the operation which is being compensated for. We can substitute different recovery policies by changing the recovery manager module.

3.7.1 Current Status

The Recovery Manager with multilevel objects and a sophisticated buffer manager will be incorporated in the next version of the simulator.

References

[1] M. Livny, "*DeNet Users Guide*", version 1.5, Dept. Comp. Science, Univ. of Wisconsin, Madison, WI 1990.

Appendix I

Parameter Declaration and Sample Values

DBManager Parameter Declaration

```
numberOfClasses : INTEGER ;
numInstances    : ARRAY [1..MAXOBJCLASSES] OF INTEGER;
numMethods      : ARRAY [1..MAXOBJCLASSES] OF INTEGER;
numSubObjects   : ARRAY [1..MAXOBJCLASSES] OF INTEGER;
numPages        : ARRAY [1..MAXOBJCLASSES] OF INTEGER;
dataContention  : ARRAY [1..MAXOBJCLASSES] OF REAL;
                (* probabilities to tune the data contention *)
methodProbability : ARRAY [1..MAXOBJCLASSES], [1..MAXNUMMETHODS] OF
                                  REAL;
                     (* probabilities of various methods *)
methodCPUTime     : ARRAY [1..MAXOBJCLASSES], [1..MAXNUMMETHODS] OF
                           REAL;
numSubObjAccessed : ARRAY [1..MAXOBJCLASSES], [1..MAXNUMMETHODS] OF
                           INTEGER;
updateProb        : ARRAY [1..MAXOBJCLASSES], [1..MAXNUMMETHODS] OF
                          REAL;
updateCpuTime      : REAL ;
replicaSite : ARRAY [1..MAXOBJCLASSES],[1..MAXNUMSITES] OF BOOLEAN;
```

```
Sample Parameter Values
-----------------------

7 := { 1 { 200 20 30 40 50 } { 1 4 4 4 4 } { 1 5 10 10 10 } { 1 5 10 10 10 }
{0.0 0.5 0.5 0.5 0.5 } { {1.0} {0.25 0.25 0.25 0.25} {0.25 0.25 0.25 0.25}
{0.25 0.25 0.25 0.25} {0.25 0.25 0.25 0.25} }
{ { 3.0 } { 1.0 1.0 1.0 1.0 } {1.0 1.0 1.0 1.0 } { 1.0 1.0 1.0 1.0 }
{ 1.0 1.0 1.0 1.0 } } { { 1 } { 1 2 3 4 } { 1 2 3 4 } { 1 2 3 4 } { 1 2 3 4 } }
{ { 1.0 } { 0.1 0.1 0.1 0.1 } { 0.1 0.1 0.1 0.1 } { 0.1 0.1 0.1 0.1 }
{ 0.1 0.1 0.1 0.1 } }   0.0   { { TRUE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE } { TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE } { TRUE FALSE FALSE
FALSE FALSE FALSE FALSE FALSE } { TRUE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE } { TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE } } };
```

Source Parameter Declaration

```
   Inperiodic     : BOOLEAN;
(* TRUE for periodic Xactions and FALSE for Aperiodic ones *)
   interArrival   : REAL;
(* This parameter is the interarrival time for
   aperiodic Xactions and period for periodic Xactions *)
   InhardFirmSoft : INTEGER;
(* This parameter says what kind of Xactions the source
   generates.
   0 - hard
   1 - firm
   2 - soft *)
   closed         : BOOLEAN;
(* This parameter specifies if the stream is closed queue or
open queue model  *)
   classExtEvents : ARRAY [1..MAXTRANSCLASS] OF INTEGER;
   classPriority  : ARRAY [1..MAXTRANSCLASS] OF INTEGER;
   arrivalDistribution : INTEGER;
(* arrivalDistribution decides the distribution
   of the arrival of each  source.
   0 - Exponential
   1 - Uniform
   2 - Binomial *)
   deadLineRatio  : INTEGER;
(* a multiplier used to control the deadline value *)
(* umass end *)
   debug          : BOOLEAN ;
   numOfTerminals : INTEGER;
   maxTransPending: INTEGER;
   thinkMean      : REAL;
   numXactClasses : INTEGER;
   dedicated      : BOOLEAN ;
   minSlack       : ARRAY [1..MAXTRANSCLASS] OF REAL;
   maxSlack       : ARRAY [1..MAXTRANSCLASS] OF REAL;
   classProb      : ARRAY [1..MAXTRANSCLASS] OF REAL ;
   (* For Validation  we are using this as meanlength*)
   maxClassNObjs  : ARRAY [1..MAXTRANSCLASS] OF INTEGER ;
   minClassNObjs  : ARRAY [1..MAXTRANSCLASS] OF INTEGER ;
   classTouchAll  : ARRAY [1..MAXTRANSCLASS] OF BOOLEAN ;
   classCpuPerRec : ARRAY [1..MAXTRANSCLASS] OF REAL ;
   classSequential: ARRAY [1..MAXTRANSCLASS] OF BOOLEAN ;

   (* All the following parameters are arrays of [class X partitions].
```

```
   There are a maximum of 8 classes and 64 partitions *)

   classAccess    : ARRAY [1..MAXTRANSCLASS],[1..MAXOBJCLASSES] OF REAL ;
 (* probability information *)
   classMeanSize  : ARRAY [1..MAXTRANSCLASS],[1..MAXOBJCLASSES] OF INTEGER ;
   sequentialAcc  : ARRAY [1..MAXTRANSCLASS],[1..MAXOBJCLASSES] OF BOOLEAN ;
   classWriteProb : ARRAY [1..MAXTRANSCLASS],[1..MAXOBJCLASSES] OF REAL ;


Sample Parameter Values
-----------------------

2 := { FALSE   250.0   2   FALSE   { 0 1 1 1 0 0 0 0 }   { 1 1 1 1 1 1 1 1 }
1   5   FALSE   1  20000 100.0   1   FALSE   {0.5} {5.0} { 1.0 }   {15}
{ 1 }   { FALSE }   { 0.0 }   { TRUE }   { { 1.0 0.0 0.0 0.0 0.0 } }
{ { 10 10 10 10 10 } }   { { TRUE TRUE TRUE TRUE TRUE} }
{ { 0.1 0.1 0.1 0.1 0.1 } }  };
```

Rule Manager Parameter Declaration

```
   classTransEventProb  : ARRAY [1..MAXTRANSCLASSES],[1..3] OF REAL;
   classObjectEventProb : ARRAY [1..MAXOBJCLASSES] OF REAL;
   classActionProb  : ARRAY [1..MAXTRANSCLASSES] OF REAL;
   classECCouplingModeProb : ARRAY [1..MAXTRANSCLASSES],[1..3] OF REAL;
   classCACouplingModeProb : ARRAY [1..MAXTRANSCLASSES],[1..3] OF REAL;
   classCascadeEventProb : ARRAY [1..MAXTRANSCLASSES] OF REAL;
   classMaxChain   : ARRAY [1..MAXTRANSCLASSES] OF INTEGER ;
   classMaxRules   : ARRAY [1..MAXTRANSCLASSES] OF INTEGER ;
   classMultipleRuleProb : ARRAY [1..MAXTRANSCLASSES] OF REAL;
   classAvgMultipleRules : ARRAY [1..MAXTRANSCLASSES] OF INTEGER;
    (* When a multiple rule condition is true
           on the average this many rules are fired
   on the occurence of a single event *)
   tempRuleCouplingModeProb:ARRAY [1..3] OF REAL;
(* Coupling mode probabilities for
     temporal rules *)
   ShareProb : REAL;
   RestartRepeatProb : REAL;
   (*MaxChainLength : INTEGER; *)
   NumAperiodicRules : INTEGER;
     (* These parameters take care of the temporal
events and their associated rules
being activated *)
   NumPeriodicRules : INTEGER;
   MinPeriod : REAL;
   MaxPeriod : REAL;
   MeanTimeEvent        : REAL;
(* This is the mean time between two
activations of an aperiodic rule *)
(* Rule task generation parameters *)
(* We assume that no nontemporal RMT is periodic *)
   NumCheckerXactClasses  : INTEGER; (* These all have to be Read-only*)
   NumActionXactClasses   : INTEGER;
   HardFirmSoft  : ARRAY [1..MAXRMTTRANSCLASSES] OF INTEGER;
   DeadLineRatio : ARRAY [1..MAXRMTTRANSCLASSES] OF INTEGER;
   CheckerClassProb  : ARRAY [1..MAXCONDTRANSCLASSES] OF REAL;
   ActionClassProb  : ARRAY [1..MAXACTIONTRANSCLASSES] OF REAL;
   ClassPriority  : ARRAY [1..MAXRMTTRANSCLASSES] OF INTEGER;
   ClassTouchAll : ARRAY [1..MAXRMTTRANSCLASSES] OF BOOLEAN;
   MaxObjCount : ARRAY [1..MAXRMTTRANSCLASSES] OF INTEGER;
   MinObjCount : ARRAY [1..MAXRMTTRANSCLASSES] OF INTEGER;
```

```
   ObjClassAccessProb  : ARRAY [1..MAXRMTTRANSCLASSES],
[1..MAXOBJCLASSES] OF REAL;
   ClassWriteProb  : ARRAY [1.. MAXRMTTRANSCLASSES],
[1..MAXOBJCLASSES] OF REAL;
   minSlack        : ARRAY [1..MAXRMTTRANSCLASSES] OF REAL;
   maxSlack        : ARRAY [1..MAXRMTTRANSCLASSES] OF REAL;
   dLinePolicy             : INTEGER;


Sample Parameter Values
-----------------------

8 := { { {0.1 0.1 0.1} }   {0.1 0.1 0.1 0.1 0.1}   {0.1}  {{1.0 0.0 0.00}}
{{1.0 0.0 0.00}} {.66} {2} {3} {0.0} {1} {1.0 0.0 0.0}    0.0    0.0    0   0
500.0   1000.0  2000.0   1   1  {2 2}   {5 5}   {1.0}   {1.0}   {1 1 }
{FALSE FALSE}   {5 5}   {5 5}   { {1.0 0.2 0.2 0.2 0.2} {1.0 0.2 0.2 0.2 0.2} }
{ {0.0 0.1 0.1 0.1 0.1} {0.1 0.1 0.1 0.1 0.1} }  {0.5 0.5} {5.0 5.0} 1};
```

Transaction Manager Parameter Declaration

```
   transDebug     : BOOLEAN;
   rejectionPolicy : INTEGER;
(*   1--> All Eligible
     2--> NotTardy
     3--> Feasible Deadlines *)
   masterLoadIo   : REAL ;
   masterLoadCPU  : REAL ;
   cohortLoadIo   : REAL ;
   cohortLoadCPU  : REAL ;
   maxMaster      : INTEGER;
   delayRestart   : BOOLEAN ;
   logBlockSize   : INTEGER ;
   logFlushTime   : REAL    ;
   logTimeOut     : REAL    ;


Sample Parameter Values
-----------------------

3 := {  TRUE  3   0.0   0.0   0.0   0.0   2000   FALSE   1   0.0   0.0  };
```

Object Manager Parameter Declaration

```
ccDebug : BOOLEAN;
ccCPU: REAL;
```

```
Sample Parameter Values
-----------------------

6 := { TRUE 0.00 };
```

Resource Manager Parameter Declaration

```
cpuRate         : REAL ;
numberOfCpus    : INTEGER;
numberOfDisks   : INTEGER ;
diskTransRate   : REAL ;
minSeekTime     : REAL ;
maxSeekTime     : REAL ;
sendReceiveTime : REAL ;
localMessTime   : REAL ;
hitRatio        : REAL ;
```

```
Sample Parameter Values
-----------------------

4 := { 1.0 1 2 20.0 10.0 10.0 0.0 0.0 1.0 };
```

Appendix II

Important Files and Description

- *DBManager.d*: The data model.
- *Source.d*: Source (transaction generator).
- *TransM.d*: Transaction Manager.
- *RM.d*: Rule Manager.
- *ResourceM.d*: Resource Manager.
- *NetworkM.d*: Network Manager.
- *CCM.d*: Object Manager.
- *CCMWW.d*: A specific CC Algorithm.
- *DBManager.ddef*: Data structures for the DBManager module.
- *Source.ddef*: Data structures for Source (transaction generator).
- *TransM.ddef*: Data structures for Transaction Manager.
- *RM.ddef*: Data structures for Rule Manager.
- *ResourceM.ddef*: Data structures for Resource Manager.
- *NetworkM.ddef*: Data structures for Network Manager.
- *CCM.ddef*: Data structures for Object Manager.
- *principal.d*: Principal file that glues the modules.
- *topology*: Topology file that sets up the interconnections and parameters of different modules.
- *Makefile*: Make file to make the simulator