Towards Model based Generation of Self - Priming and Self - Checking Conformance Tests for Implementing Atomic Read/Write Shared Memory in Mobile Ad hoc Networks

Fatma.A. Omara[1] and Reham.A.Shihata[2],

[1]Computer science department, Cairo University,
Information Systems and Computers Faculty, Cairo-Egypt.University.
[2] Math's Department. Science Faculty,
EL Minufiya University, Shebin-El Kom -Egypt.

Summary
This paper describes a model-based approach to generate conformance tests for interactive applications .this method address generation of small yet effective set of test frames for testing individual operations, then a set up sequence that brings the system under test in an appropriate state for a test frame (self – priming), also a verification sequence for expected output and state changes (self – checking), finally, negative test cases in the presence of exceptions. This method exploits a novel mutation scheme applied to operations modeled as relationships among parameters and state variables; a set of novel abstraction techniques which result in a compact Finite State Automaton (FSA); and search techniques to automatically generate the set up and verification sequences. We illustrate this method with a geoquorum approach for implementing atomic read/write shared memory in mobile ad hoc networks as a simple application.

Key words:
software engineering lifecycle, conformance test, specification phase, FSA, mobile ad hoc networks

1. Introduction

In this paper, we use a model-based approach to generate conformance tests for the geoquorum approach which is considered the application of this paper .This model is proposed to that specify some contributions which are:

1. Abstraction techniques for extracting FSA. And show how to extract a compact FSA from a test specification. The abstraction techniques include variable hiding which eliminates certain system state variables from the state labels and state merging which merges two states with different labels but same transitions [1].
2. Generation of verification scripts, the issue is addressed of verification scripts by exploiting a common characteristics of several interactive systems: availability of reader operations a reader operation propagates certain aspects of the system state information to the external environment.
3. Generation of negative test cases, the negative test cases are generated for an operation in presence of exceptional results by exploiting the state updates of the operations successful result [2]. Conformance test generation techniques based on FSA representation of operations in communication protocols have been reported in [2].

Verification sequences for the control part of a protocol entity use one of these: First, Distinguishing sequence which produces a district output when applied to every state in FSA. Second, A unique Input/Output sequence for each state S in the FSA, which distinguishes S from all other states in the FSA is determined. Third, a characterization set which consists of operation sequences such that the last output observed from the application of these sequences is different at each state of the FSA.

2. The Geoquorum-Approach(the application)

In this paper the Geoquorum algorithm is presented for implementing the atomic read/write in shared memory of mobile ad hoc networks. This approach is based on associating abstract atomic objects with certain geographic locations. It is assumed that the existence of Focal Points, geographic areas that are normally "populated" by mobile nodes. For example: a focal point may be a road Junction, a scenic observation point. Mobile nodes that happen to populate a focal point participate in implementing a shared atomic object, using a replicated state machine approach. These objects, which are called focal point objects, are prone to Occasional failures when the corresponding geographic areas are depopulated. The Geoquorums algorithm uses the fault-prone focal point objects to implement atomic read/write operations on a fault-tolerant

Manuscript received April 5, 2010
Manuscript revised April 20, 2010

virtual shared object. The Geoquorums algorithm uses a quorum- based strategy in which each quorum consists of a set of focal point objects. The quorums are used to maintain the consistency of the shared memory and to tolerate limited failures of the focal point objects, which may be caused by depopulation of the corresponding geographic areas. The mechanism for changing the set of quorums has presented, thus improving efficiency [4]. Overall, the new Geoquorums algorithm efficiently implements read/write operations in a highly dynamic, mobile network. In this chapter, a new approach to designing algorithms for mobile ad hoc networks is presented. An ad hoc network uses no pre-existing infrastructure, unlike cellular networks that depend on fixed, wired base stations. Instead, the network is formed by the mobile nodes themselves, which co-operate to route communication from sources to destinations. Ad hoc communication networks are by nature, highly dynamic. Mobile nodes are often small devices with limited energy that spontaneously join and leave the network. As a mobile node moves, the set of neighbors with which at can directly communicate may change completely. The nature of ad hoc networks makes it challenging to solve the standard problems encountered in mobile computing, such as location management using classical tools. The difficulties arise from the lack of a fixed infrastructure to serve as the backbone of the network. In this chapter developing a new approach that allows existing distributed algorithm to be adapted for highly dynamic ad hoc environments one such fundamental problem in distributed computing is implementing atomic read/ write shared memory [5]. Atomic memory is a basic service that facilitates the implementation of many higher level algorithms. For example: one might construct a location service by requiring each mobile node to periodically write its current location to the memory. Alternatively, a shared memory could be used to collect real – time statistics, for example: recording the number of people in a building here, a new algorithm for atomic multi writes/multi- reads memory in mobile ad hoc networks. The problem of implementing atomic read/write memory is divided into two parts; first, we define a static system model, the focal point object model that associates abstract objects with certain fixed geographic locales. The mobile nodes implement this model using a replicated state machine approach. In this way, the dynamic nature of the ad hoc network is masked by a static model. Moreover it should be noted that this approach can be applied to any dynamic network that has a geographic basis. Second, an algorithm is presented to implement read/write atomic memory using the focal point object model. The implementation of the focal point object model depends on a set of physical regions, known as focal points .The mobile nodes within a focal point cooperate to simulate a single virtual object, known as a focal point object. Each focal point supports a local broadcast service, LBcast which provides reliable, totally ordered broadcast. This service allows each node in the focal point to communicate reliably with every other node in the focal point. The focal broadcast service is used to implement a type of replicated state machine, one that tolerates joins and leaves of mobile nodes. If a focal point becomes depopulated, then the associated focal point object fails. (Note that it doesn't matter how a focal point becomes depopulated, be it as a result of mobile nodes failing, leaving the area, going to sleep. etc. Any depopulation results in the focal point failing). The Geoquorums algorithm implements an atomic read/write memory algorithm on top of the geographic abstraction, that is, on top of the focal point object model. Nodes implementing the atomic memory use a Geocast service to communicate with the focal point objects. In order to achieve fault tolerance and availability, the algorithm replicates the read/write shared memory at a number of focal point objects. In order to maintain consistency, accessing the shared memory requires updating certain sets of focal points known as quorums. An important aspect of our approach is that the members of our quorums are focal point objects, not mobile nodes. The algorithm uses two sets of quorums (I) get-quorums (II) put- quorums with property that every get-quorum intersects every put-quorum. There is no requirement that put-quorums intersect other put-quorums, or get-quorums intersect other get-quorums. The use of quorums allows the algorithm to tolerate the failure of a limited number of focal point objects. Our algorithm uses a Global Position System (GPS) time service, allowing it to process write operations using a single phase, prior single-phase write algorithm made other strong assumptions, for example: relying either on synchrony or single writers. This algorithm guarantees that all read operations complete within two phases, but allows for some reads to be completed using a single phase: the atomic memory algorithm flags the completion of a previous read or write operation to avoid using additional phases, and propagates this information to various focal paint objects[4]. As far as we know, this is an improvement on previous quorum based algorithms. For performance reasons, at different times it may be desirable to use different times it may be desirable to use different sets of get quorums and put-quorums. For example: during intervals when there are many more read operations than write operations, it may be preferable to use smaller get-quorums that are well distributed, and larger put-quorums that are sparsely distributed. In this case a client can rapidly communicate with a get-quorum while communicating with a put – quorum may be slow. If the operational statistics change, it may be useful to reverse

the situation. The algorithm presented here includes a limited "reconfiguration" Capability: it can switch between a finite number of predetermined quorum systems, thus changing the available put-quorums and get –quorums. As a result of the static underlying focal point object model, in which focal point objects neither join nor leave, it isn't a severe limitation to require the number of predetermined quorum systems to be finite (and small). The resulting reconfiguration algorithm, however, is quite efficient compared to prior reconfigurable atomic memory algorithms. Reconfiguration doesn't significantly delay read or write operations, and as no consensus service is required, reconfiguration terminates rapidly [5].

The mathematical notation for the geoquorums approach

- I the totally- ordered set of node identifiers.
- $I_0 \in I$, a distinguished node identifier in I that is smaller than all order identifiers in I.
- S, the set of port identifiers, defined as $N^{>0} \times OP \times I$,
  Where OP= {get, put, confirm, recon- done}.
- O, the totally- ordered, finite set of focal point identifiers.
- T, the set of tags defined as $R^{\geq 0} \times I$.
- U, the set of operation identifiers, defined as $R^{\geq 0} \times S$.
- X, the set of memory locations for each $x \in X$:
  - V_x the set of values for x
  - $v_{0,x} \in V_x$, the initial value of X
- M, a totally-ordered set of configuration names
- $c_0 \in M$, a distinguished configuration in M that is smaller than all Other names in M.
- C, totally- ordered set of configuration identifies, as defined as: $R^{\geq 0} \times I \times M$
- L, set of locations in the plane, defined as $R \times R$

Fig .1 Notations Used in The Geoquorums Algorithm.

Variable Types for Atomic Read/Write object in Geoquorum Approach
for Mobile Ad Hoc Network

The specification of a variable type for a read/write object in geoquorum approach for mobile ad hoc network is presented. A read/write object has the following variable type (see fig .2) [4].

Put/get variable type τ
State
Tag $\in$T, initially< $0.i_0$>
Value $\in$V, initially v_0
Config-id $\in$C, initially< $0, i_0, c_0$>
Confirmed-set $\subseteq$ T, initially Ø
Recon-ip, a Boolean, initially false
Operations
Put (new-tag, new-value, new-Config-id)
If (new-tag> tag) then
Value ←new-value
Tag ← new-tag
If (new-Config-id > Config-id) then
Config-id ← new-config-id
Recon-ip ← true
Return put-Ack (Config-id, recon-ip)
Get (new-config-id)
If (new-config-id >Config-id) then
Config-id ← new-Config-id
Recon-ip ←true
Confirmed ← (tag $\in$confirmed-set)
Return get-ack (tag, value, confirmed, Config-id, recon-ip)
Confirm (new-tag)
Confirmed-set ←confirmed –set U {new-tag}
Return confirm-Ack
Recon –done (new-Config-id)
If (new-Config-id=Config-id) then
Recon-ip ←false
Return recon-done-Ack ( )

Fig .2 Definition of the Put/Get Variable Type τ

2.1 Operation Manager

In this section the Operation Manger (OM) is presented, an algorithm built on the focal/point object Model. As the focal point Object Model contains two entities, focal point objects and Mobile nodes, two specifications is presented , on for the objects and one for the application running on the mobile nodes [5] .

(A) Operation Manager Client

This automaton receives read, write, and recon requests from clients and manages quorum accesses to implement these operations (see fig .3). The Operation Manager (OM) is the collection of all the operation manager clients (OM_i, for all i in I).it is composed of the focal point objects, each of which is an atomic object with the put/get variable type:

Operation Manager Client Transitions
Input write $(Val)_i$
Effect:
Current-port-number ←
Current-port-number +1
Op ← < write, put, <clock, i>, Val, recon-ip, <0, i0, c0>, Ø>
Output write-Ack $( )_i$
Precondition:
Conf-id=<time-stamp, Pid, c>
If op .recon-ip then
$\surd C' \in M, \exists P \in$ put-quorums(C'): P $\subseteq$ op. acc
Else

∋ P ∈ put-quorums(C): P ⊆ Op. acc
Op .phase=put
Op. type=write
Effect:
Op. phase ← idle
Confirmed ← confirmed U {op. tag}
Input read ( )$_i$
Effect:
Current-port-number ←
Current-port-number +1
Op ← < read, get, ⊥, ⊤, recon-ip, <0, i0, c0>, Ø>
Output read-ack (v)$_i$
Precondition:
Conf-id=<time-stamp, Pid, c>
If op. recon-ip then
√ C′ ∈ M, ∋ G ∈ get-quorums(C′): G ⊆ op. acc
Else
∋ G ∈ get-quorums(C): G ⊆ op. acc
Op. phase=get
Op. type=read
Op. tag ∈ confirmed
v= op. value
Effect:
Op .phase ← idle
Internal read-2( )$_i$
Precondition:
Conf-id=<time-stamp, Pid, c>
√ C′ ∈ M, ∋ G ∈ get-quorums(C′): G ⊆ op. acc
Else
∋ G ∈ get-quorums(C): G ⊆ op. acc
Op. phase=get
Op. type=read
Op. tag ∉ confirmed
Effect:
Current-port-number ←
Current-port-number +1
Op. phase ← put
Op. Recon. ip ← recon-ip
Op. acc ← Ø
Output read-Ack (v)$_i$
Precondition:
Conf-id=<time-stamp, Pid, c>
If op. recon-ip then
√ C′ ∈ M, ∋ P ∈ put-quorums(C′): P ⊆ op. acc
Else
∋ P ∈ put-quorums(C): P ⊆ op. acc
Op. phase=put
Op. type=read
v=op. value
Effect:
Op. phase ← idle
Confirmed ← confirmed U {op. tag}
Input recon (conf-name)$_i$
Effect:
Conf-id ← <clock, i, conf-name>
Recon-ip ← true
Current-port-number ←
Current-port-number +1
Op ← < recon, get, ⊥, ⊥, true, conf-id, Ø>
Internal recon-2(cid)$_i$
Precondition
√ C′ ∈ M, ∋ G ∈ get-quorums(C′): G ⊆ op. acc
√ C′ ∈ M, ∋ P ∈ put-quorums(C′): P ⊆ op. acc
Op. type=recon
Op. phase=get
Cid=op. recon-conf-id
Effect
Current-port-number ←
Current-port-number +1
Op. phase ← put
Op. acc ← Ø
Output recon-Ack(c)$_i$
Precondition
Cid=op. recon-conf-id
Cid= <time-stamp, Pid, c>
∋ P ∈ put-quorums(C): P ⊆ op. acc
Op. type=recon
Op. phase=put
Effect:
If (conf-id=op. recon-conf-id) then
Recon-ip ← false
Op. phase ← idle
Input geo-update (t, L)$_i$
Effect:
Clock ← 1

Fig .3 Operation Manager Client Read/Write/Recon and Geo-update Transitions for Node

2.2 Focal Point Emulator Overview

The focal point emulator implements the focal point object Model in an ad hoc mobile network. The nodes in a focal point (i.e. in the specified physical region) collaborate to implement a focal point object. They take advantage of the powerful LBcast service to implement a replicated state machine that tolerates nodes continually joining and leaving .This replicated state machine consistently maintains the state of the atomic object, ensuring that the invocations are performed in a consistent order at every mobile node [4].In this section an algorithm is presented to implement the focal point object model. the algorithm allows mobile nodes moving in and out of focal points, communicating with distributed clients through the geocast service, to implement an atomic object (with port set q=s)corresponding to a particular focal point. We refer to this algorithm as the Focal Point Emulator (FPE). The FPE client has three basic purposes. First, it ensures that

each invocation receives at most one response (eliminating duplicates).Second, it abstracts away the geocast communication, providing a simple invoke/respond interface to the mobile node [5]. Third, it provides each mobile node with multiple ports to the focal point object; the number of ports depends on the atomic object being implemented. The remaining code for the FPE server is in fig .4.When a node enters the focal point, it broadcasts a join-request message using the LBcast service and waits for a response. The other nodes in the focal point respond to a join-request by sending the current state of the simulated object using the LBcast service. As an optimization, to avoid unnecessary message traffic and collisions, if a node observes that someone else has already responded to a join-request, and then it does not respond. Once a node has received the response to its join-request, then it starts participating in the simulation, by becoming active. When a node receives a Geocast message containing an operation invocation, it resends it with the Lbcast service to the focal point, thus causing the invocation to become ordered with respect to the other LBcast messages (which are join-request messages, responses to join requests, and operation invocations).since it is possible that a Geocast is received by more than one node in the focal point ,there is some bookkeeping to make sure that only one copy of the same invocation is actually processed by the nodes. There exists an optimization that if a node observes that an invocation has already been sent with LBcast service, then it does not do so. Active nodes keep track of operation invocations in the order in which they receive them over the LBcast service. Duplicates are discarded using the unique operation ids. The operations are performed on the simulated state in order. After each one, a Geocast is sent back to the invoking node with the response. Operations complete when the invoking node with the response. Operations complete when the invoking node remains in the same region as when it sent the invocation, allowing the geocast to find it. When a node leaves the focal point, it re-initializes its variables .A subtle point is to decide when a node should start collecting invocations to be applied to its replica of the object state. A node receives a snapshot of the state when it joins. However by the time the snapshot is received, it might be out of date, since there may have been some intervening messages from the LBcast service that have been received since the snapshot was sent. Therefore the joining node must record all the operation invocations that are broadcast after its join request was broadcast but before it received the snapshot .this is accomplished by having the joining node enter a "listening" state once it receives its own join request message; all invocations received when a node is in either the listening or the active state are recorded, and actual processing of the invocations can start once the node has received the snapshot and has the active status. A precondition for performing most of these actions that the node is in the relevant focal point. This property is covered in most cases by the integrity requirements of the LBcast and Geocast services, which imply that these actions only happen when the node is in the appropriate focal point.

Focal Point Emulator Server Transitions

Internal join ( ) $_{Obj\ ,\ i}$
Precondition:
Location ∈ FP-location
Status=idle
Effect:
Join-id ←<clock, i>
Status← joining
Enqueue (Lbcast-queue, <join-req, join-id>)
Input Lbcast- rcv (< join-req, jid>) $_{obj,\ i}$
Effect:
If ((status=joining)) ∧ (jid=Join-id)) then
Status ←listening
If ((status=active)) ^ jid ∉ answered-join-reqs)) then
Enqueue (LBcast-queue, < join-ack, jid, val>)
Input Lbcast- rcv (<join-ack, jid, v>) $_{obj,\ i}$
Effect:
Answered-join-reqs ← answered-join-reqs U {jid}
If ((status=listening) ∧ (jid =join-id)) then
Status ← active
val ← V
Input Geocast –rcv (< invoke, inv, oid, loc, FP-loc>) $_{obj,i}$
Effect:
If (FP-loc=FP-location) then
If (<inv, oid, loc> ∉ pending-ops U completed ops) then
Enqueue (Lbcast-queue, <invoke, inv, oid, loc>)
Input LBcast –rcv (< invoke, inv, oid, loc>) $_{obj,i}$
Effect:
If ((status=listening V active) ∧
(<inv, oid, loc> ∉ pending-ops U completed-ops)) Then
Enqueue (pending-ops, <inv, oid, loc>)
Internal simulate-op (inv) $_{obj,\ i}$
Precondition:
Status=active
Peek (pending-ops) =<inv, oid, loc>
Effect:
(Val, resp)← δ (inv, val)
Enqueue (geocost- queue, < response, resp, oid, loc>)
Enqueue (completed-ops, Dequeue (pending-ops))
Internal leave ( ) $_{obj,\ i}$
Precondition:
Location ∉ fp-location
Status ≠ idle
Effect:
Status← idle
Join-id← <0, i_0>
Val ← v_0

Answered -join- reqs← Ø
Pending –ops ← Ø
Completed-ops ← Ø
Lbcast-queue ← Ø
Geocast-queue ← Ø
Output Lbcast (m) $_{obj, i}$
Precondition:
Peek (Lbcast-queue) =m
Effect:
Duqueue (Lbcast- queue)
Output geocast (m) $_{obj, i}$
Precondition:
Peek (geocast-queue) =m
Effect:
Dequeue (geocost- queue)
Input get-update (l, t) $_{obj,i}$
Effect:
Location ← l
Clock← t

Fig. 4 FPE server transitions for client i and object Obj of variable type τ = <V, v_0, invocations, responses, δ >

The illustration of the above terminology and motivate the testing method with the GeoQuorum Algorithm for implementing atomic read/write shared memory in mobile ad hoc networks:

<Component> Mobile ad hoc network
Port <current- port-number, confirm, i>
T tag
nv new-value
nt new-tag
C config-id
nc new-config-id
re recon-ip
< /Component >
<FUNCTION> put invocation
In pin =
{New-value, new-tag, config-id}
Out rc =new-config – id
REQUIRES (new-tag >tag, new-config–id > config–id
EXCEPTION
(Recon – ip = false).
RESULT
Put-invocation = put-Ack–response
< DATA>
New-value = {INT}
New-tag = {INT}
New-config-id = {INT}
</DATA>
</FUNCTION>
<FUNCTION> get-invocation
In amt = {config-id}
Out rc = {new-config-id}
REQUIRES (new-config-id> config-id)
EXCEPTION
(Tag > new-tag)
RESULT
Get-invocation = get-Ack response
< DATA>
Tag, new-tag = {INT}
Config-id, new-config – id = {INT}
</DATA>
<FUNCTION>
<FUNCTION> confirm – invocation
In nt = {new-tag, confimed-set}
Out cs = {new –confirmed-set}
REQUIRES(new-confirmed-set= (confirmed-set) ∪ {new-tag})
EXCEPTION
(Confirmed-set > new-confirmed-set)
RESULT
Confirm-invocation=confirm-Ack-response
<DATA>
New-tag = {INT}
Confirmed – set, new-confirmed-set = {INT}
</DATA>
<FUNCTION>
<FUNCTION>
Recon-done invocation
In nm = {config-id, recon-ip}
Out bs = {new-config-id, op. recon-conf-id}
REQUIRES ((new-config-id =config-id) ∨ recon-ip = true).
EXCEPTION (conf -id < > op. recon-conf-id →recon-ip = false)
RESULT
Recon-done-invocation = recon-done-Ack
<DATA>
Conf-id, new-conf-id = {INT}
Op-recon-conf-id = {INT}
Recon-ip = {Boolean}
</FUNCTION>
<SUCCESS>
Put-Ack-response = Okay
Get-Ack- response = Okay
Confirm-Ack-response = Okay
Recon-done-Ack = Okay
</SUCCESS>

Fig.5 Functional Description for implementing atomic read/write shared memory in mobile ad hoc network application

The Values for Specification and Abstraction Language for Testing

F = {put-invocation, get-invocation, confirm-invocation, recon-done- invocation}
φ= { t, nv, nt, c, nc,re}
R_{put} = {rc}

ϑ_{Put} = {put – ack – response}
R_{Get} = {rc}
ϑ_{Get} = {get – ack – response}
$R_{confirm}$ = {cs}
$\vartheta_{Confirm}$ = {confirm – ack – response}
$R_{recon\text{-}done}$ = {bs}
$\vartheta_{Recon\text{-}done}$ = {recon- done – ack}
D_{pin} = invalid ∪ valid
V_{pin}^{Put} = {integer}
D_{pin}^{Put} = invalid ∪ valid
V_{pin}^{Get} = {integer}
D_{pin}^{Get} = invalid ∪ valid
$V_{pin}^{Confirm}$ = {integer}
$D_{pin}^{Confirm}$ = invalid ∪ valid
$V_{pin}^{Recon\text{-}done}$ = {integer}
φ_{init} = {t = 0, nv = 1, nt =1, c=1, nc = 1, re =1}

3. Analysis and Test Generation

3.1 Issues in Conformance Testing:

For testing a software system which implements atomic object read/write shared memory in mobile ad hoc networks specification manually, a test designer enumerates relevant conformance issues *Table. 1* illustrates some example conformance issue for the mobile ad hoc network application. Each row in table poses a conformance issue in the form of a question and a sample test case.

Table.1 Example Conformance Questions and a Sample Target Test Case

Row	Conformance issue	A sample (Partial) Test Case
1	Does new-tag smaller than tag?	new-tag > tag
2	Does new config-id smaller than config-id?	new-config-id > config-id
3	Can put-invocation be put-Ack-response?	put-invocation=put-Ack-response
4	Can get-invocation be get- Ack -response?	get-invocation=get-Ack-response
5	Can confirm- invocation be confirm-Ack-response?	confirm-invocation=confirm-Ack-response
6	Can recon-done invocation be recon-done-Ack?	recon-done-invocation=recon-done-Ack

3.2 Derivation of Test Frames:

In this paper, the important issue is automatically generating test cases which address the conformance issues of interest based only on the specification furnished in fig.6 from the foregoing discussion about Table.1; this figure explains all nodes of the related work can be tested. It is evident that derivation of test frames is crucial step of the test generation process; we select test frames based on the intuition that each test frame should target exactly one interesting aspect of the operations behavior [6].

3.3 FSA Extraction:

An enumerated state transition diagram for implementing atomic read/write shared memory in mobile ad hoc networks where each state is labeled by valuation of each context variable has an infinite number of states, since values can be unbounded with an infinite string of successive invocations of the deposit operation. To prune the number of states in the transition diagram, we employ three kinds of abstractions: Variable Hiding: The variable hiding is used to eliminate certain context variables from the state label. We use the notion of Absolute partitions, which are independent of other variables, to identify the subset of context variables relevant for labeling the states [7].Abstract Interpretation: Even after eliminating the irrelevant context variables, the resulting state diagram could contain states which are not meaningful from testing perspective. This is a consequence of using variable values as state labels [7]. State Merging: The state labels in the FSA of Fig.7 consist of a predicate on each relevant variable. However, not all predicates are relevant in all states we exploit this observation in our last abstraction technique, called states merging, to collapse a pair of states which differ only in the predicates of variables which are independent in all the operations eligible in both states. Fig.7 shows the final FSA for implementing atomic read/write shared memory in mobile ad hoc network after all of our abstraction techniques have been employed with the important states of the application with out any emulators.

3.4 FSA Traversal:

After constructing the abstract FSA for the specification, we use the techniques developed to test FSAs to guide the test generation. Specifically we used the algorithm to complete the transition tour of the abstract FSA from

adapted to account for nondeterministic and infeasible transitions in the FSA [8].

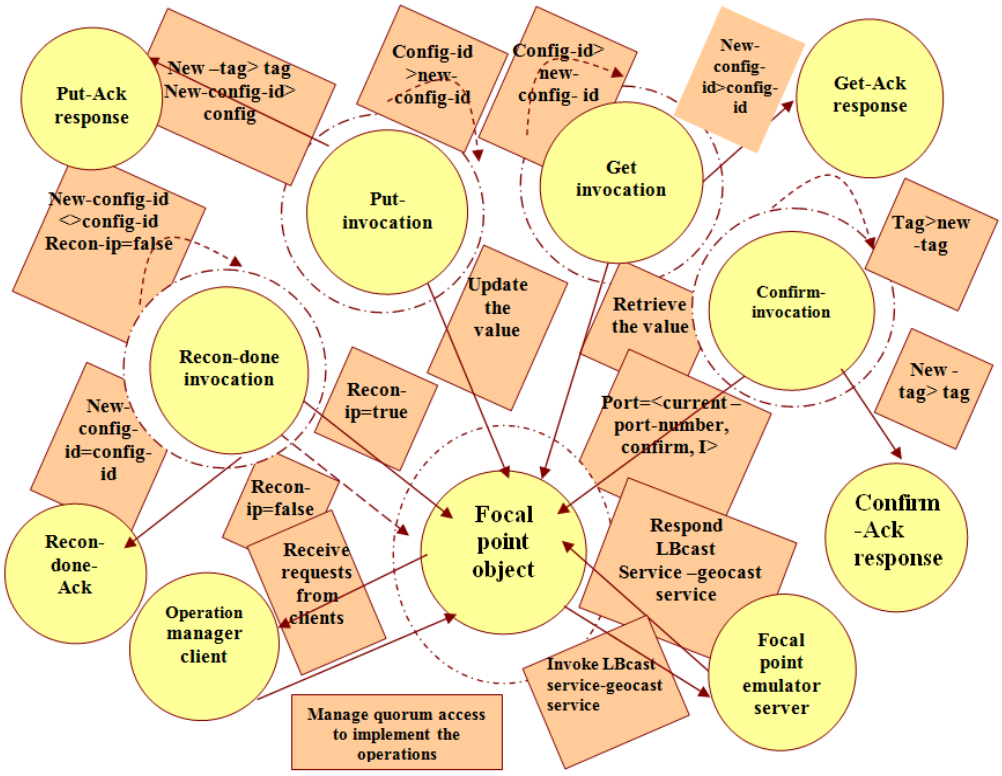

Fig.6. an FSA for Atomic Read/Write Shared Memory in Mobile Ad Hoc Networks

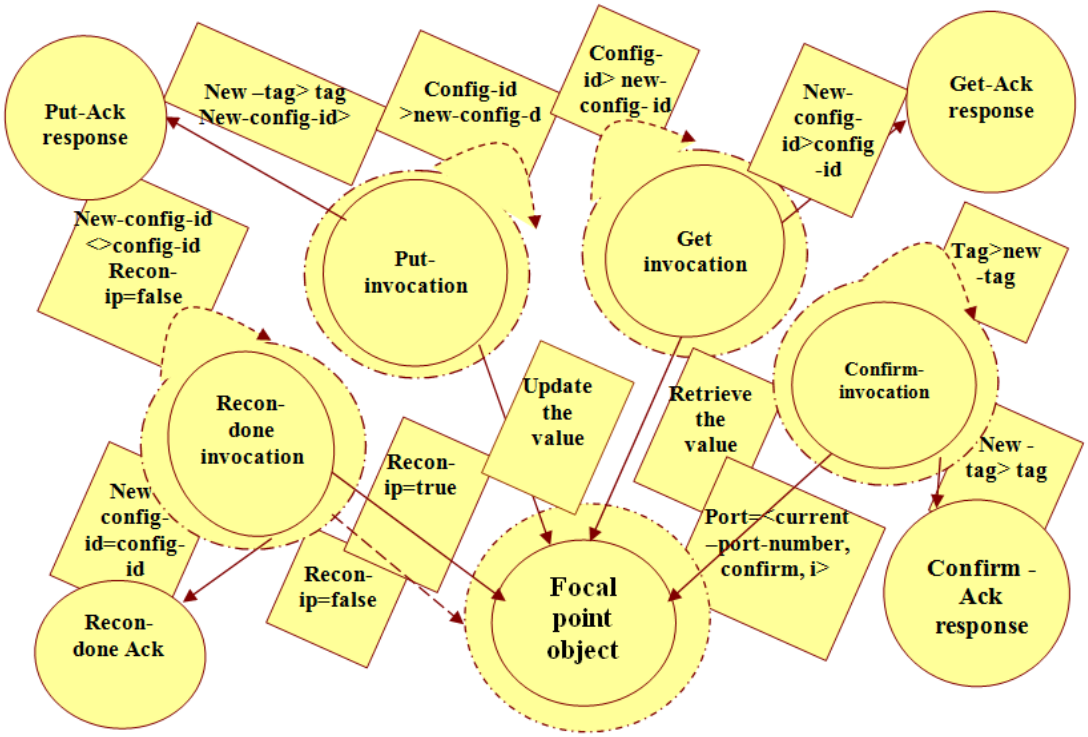

Fig.7 Final FSA for Atomic Read/Write Shared Memory in Mobile Ad Hoc Networks

3.5 Generation of Negative Tests:

To generate negative tests cases, we employ a novel variation of the mutation theme. In traditional mutation based approach to test generation, the target is mutated through deletion or modification of the information in the target [7] [9]. In this approach, inserting information in the target thus inserting the actions associated with a normal result of an operation into the state updates of its exceptional results. Then regenerating the abstract FSA and attempt to identify distinguishing sequences as before, in the next figures we separate the basic states in fig.7 into four states with explanations of their conditions which satisfied and not satisfied as explained in (Fig. 8, 9, 10, 11).

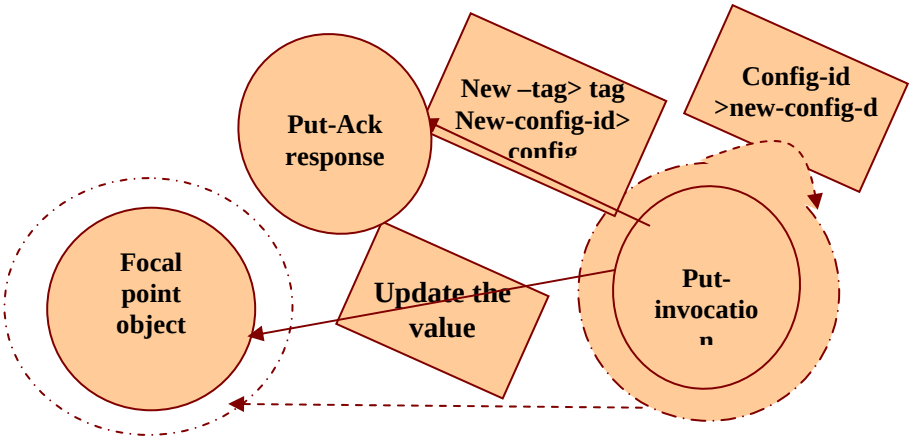

Fig.8Mutated- configurations for put- invocation in atomic read/write shared Memory in mobile ad hoc networks

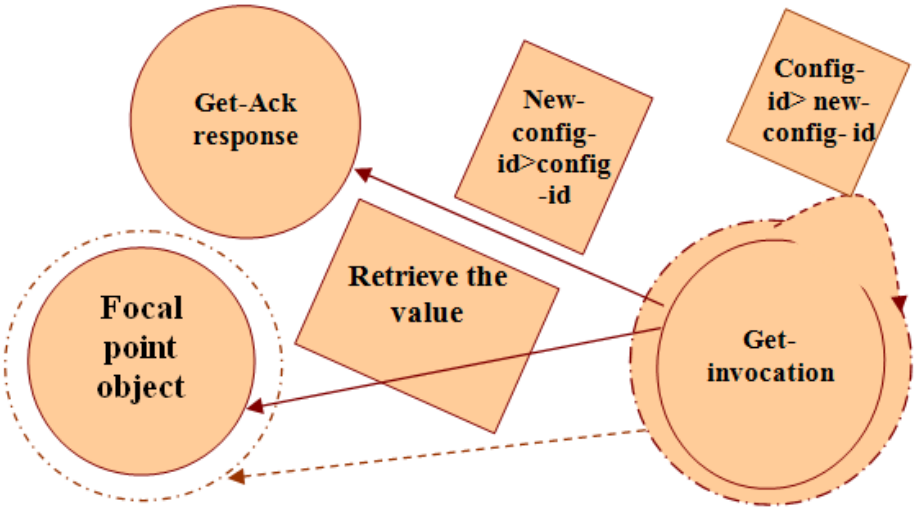

Fig.9 Mutated- configurations for get- invocation in atomic read/write shared memory in mobile ad hoc networks

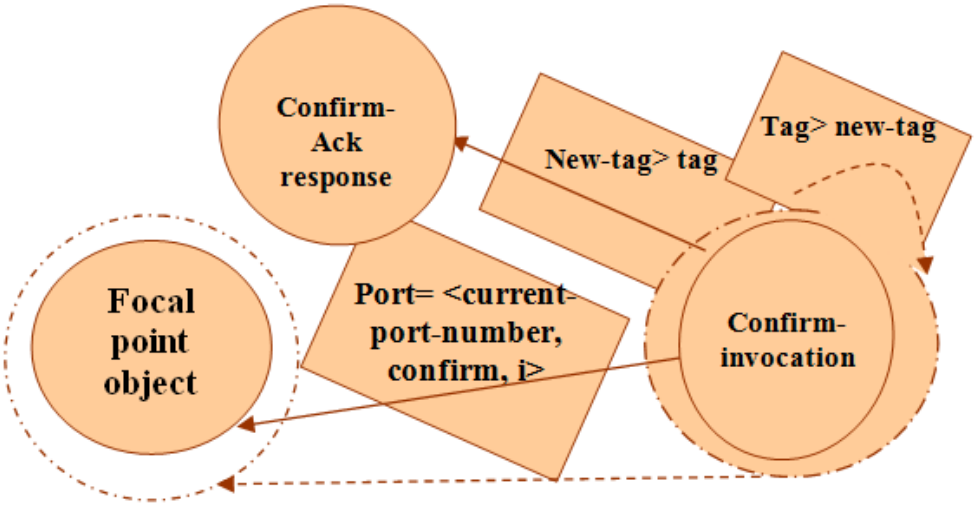

Fig.10 Mutated- configurations for confirm- invocation in atomic read/write shared memory in mobile ad hoc networks

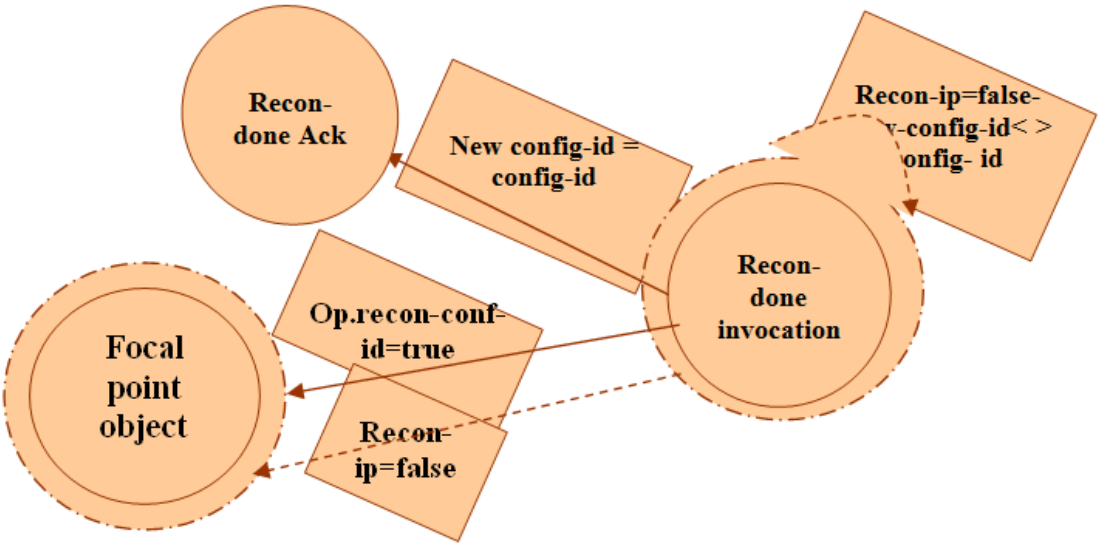

Fig.11 Mutated- configurations for recon-done invocation in atomic read/write shared memory in mobile ad hoc networks

4. Issues in Optimization

4.1 Free Variables

An operation specification may contain free parameters which don't participate in any guard. In the absence of any prior information about interactions among partitions of free parameters, one test variation with particular partition value for a free parameter is as good as any other. We adopt this philosophy to minimize the number of generated test variations. The test generation algorithm ensures that all partitions of free parameters are covered at least once [9].

4.2 Minimization of Data Values

The test designer specifies a finite number of data values to be used for partition of input parameters. This method guarantees that each data value gets used at least once however, two issues regarding how often each data value gets used need to be addressed [10]. First, a parameter partition may be eligible for more than one result of the operation: should each value for the parameter get used for each result?. Second, a given result may be eligible in more than one configuration with the same parameter partitions: should each value for the partition get used to produce the result in each configuration? In this method, distinguishing between normal and exceptional results to address both these issues, for partitions which are

applicable only for exceptional results, each value gets used at least once (not necessarily in each eligible configuration). For partitions which are applicable in both exceptional and normal results we assume that one demonstration of an abnormal result is adequate and don't ensure that each eligible value gets used. However, each eligible data value gets used in each normal result at least once (again, not necessarily in every configuration).

Conclusions and Future Work

In this paper, we have tested the implementation of atomic object read/write shared memory in mobile ad hoc network using a new method to automatically generate self-priming and self checking conformance test cases from a system model developed by the test designer. The system model consists of specification of operations in terms of relationships among parameters that they take and context variables that they manipulate. However, the mutation operators that are used in this paper are simple. We would like to conduct experiments to study their effectiveness and investigate other operators and their effect on both the test suite size and fault detection capabilities.

Acknowledgment

The authors would like to thank the Conference INFOS 2010 (Cairo University) reviewers for their constructive comments and suggestions.

References

[1] I.EL-Far, Automated Construction of Software Behavior Models, Master's Thesis, Florida Institute of Technology, Melbourne, FL, 1999.

[2] E.Farchi, A. Hartman, S.S.Pinter, "Using a Model-Based Test Generator to Test for Standard Conformance," IBM Syst.J.41 (2002)89-110.

[3] W.Greskamp, Y.Gurevich, W.shculte, M.Veanes," Generating Finite State Machines from Abstract State Machines", In: Proceedings of the 2002 International Symposium of Software Testing Analysis July (2002) 112-122.

[4] Dolev,S.,Gilbert,S.Lynch,N.A.,Shvartsman,A.A.,Welch,J.L: "Geoquorums:Implementing Atomic Memory in Mobile Ad Hoc Networks ".In: Proceeding of The 17th International Conference on Distributed Computing,PP:306-320(2003).

[5] Haas, Z.J., Liang, B.A., "Ad Hoc Mobile Management with Uniform Quorum Systems", IEEE/ACM Transactions on Networking 7(2), PP: 228-240(2002).

[6] KO, Y.B., Vaidya, N.,"A Protocol for Geocasting in Mobile Ad Hoc Networks, In: Proceedings of the IEEE International Conference on Network Protocols, PP: 240-249(2000).

[7] A.M.Memon, M.E.Pollock, M, L.Soffa,"Using a Goal-Driven Approach to Generate Test Cases for GUIS," In: Proceedings of the 21st International Conferences of Software Engineering, May, ACM Press New York, 2000, PP: 257-266.

[8] P. Froehlich, J. Link "Automated Test Generation from Dynamic Models," In: Proceedings of the ECOOP2000, LNCS 1850, Springer, Berlin, 2000, PP: 472-491.

[9] A.M. Paradkar," An Integrated to Automate Generation of Function Tests for APIS", In: Proceedings of the International Symposium of Software Reliability Engineering, 2000, October (2000) PP: 304-316.

[10] A. Paradkar," Towards Model-Based Generation of Self-Priming and Self-Checking Conformance Tests for Interactive Systems ", In: Proceedings of the Information and Software Technology 46(2004), PP: 315-322.