
Specification and Performance Evaluation for Atomic Read/ Write Shared Memory in Mobile Ad Hoc Networks Using Fuzzy Logic.

Reham.A.Shihata

**PhD in Computer Science, EL Menoufia University – Egypt. Software Consultant
in Egyptian Syndicate of Programmers & Scientists. .**

Abstract : In this paper an efficient fuzzy logic based solution for the specification and performance evaluation depending on generation of fuzzy rules is presented. A new property matching mechanism is defined; in general, a requirement with attributes is chandelled in the following manner: the basic functionality is ensured, matching properties names according to the classical reading/writing strategy. Usually several solutions result from this first step. Second, the preliminary solutions are selected and hierarchies according to the degree of attribute matching. This is done by fuzzy logic. . I will describe the basic principles of the proposed solutions and illustrate them for implementing atomic read write shared memory in mobile ad hoc network.

Keywords: Specification Analysis, Formal Specification, Performance Evaluation, Mobile Ad Hoc Network, Fuzzy Logic.

I. INTRODUCTION

A software system is viewed as a set of components that are connected to each other through connectors. A software component is an

implementation of some functionality, available under the condition of a certain contract, independently deployable and subject to composition. In the specification approach, each component has a set of logical points of interaction with its environment. The logic of a component composition (the semantic part) is enforced through the checking of component contracts. Components may be simple or composed. A simple component is the basic unit of composition that is responsible for certain behavior. Composed components introduce a grouping mechanism to create higher abstractions and may have several inputs and outputs. Components are specified by means of their provided and required properties. Properties in this specification approach are facts known about the component. A property is a name from a domain vocabulary set and may have refining sub-properties (which are also properties) or refining attributes that are typed values [1]. The component contracts specify the services provided by the component and their characteristics on one side and the obligations of client and environment components on the other side. Most often the provided services and their quality depend on the services offered by other parties, being subject to a contract. A component assembly is valid if it provides all individual components are respected. A contract for a component is respected if all its required properties have found a match. The criterion for a semantically correct component assembly is matching all required properties with provided properties on every flow in the system. In this specification approach, it is not necessary that a requirement of a component is matched by a component directly

connected to it. It is sufficient that requirements are matched by some components that are present on the flow connected to the logical point; these requirements are able to propagate [2].

II. FUZZY ATTRIBUTES

A property consists of a name describing functionality and attributes that are either type's values or fuzzy terms. The names used for the properties and for the attributes are established through a domain-specific vocabulary .Such a restriction is necessary because a totally free-text specification makes the retrieval difficult, producing false- positive or false-negative matching due to the use of a non-standard terminology[2]. In this work, the domain specific vocabulary must also describe the domains of the fuzzy attributes (linguistic variables) for each property as well as the membership functions for the fuzzy terms. The membership functions for all linguistic variables are considered of triangular shape as in this fig.1

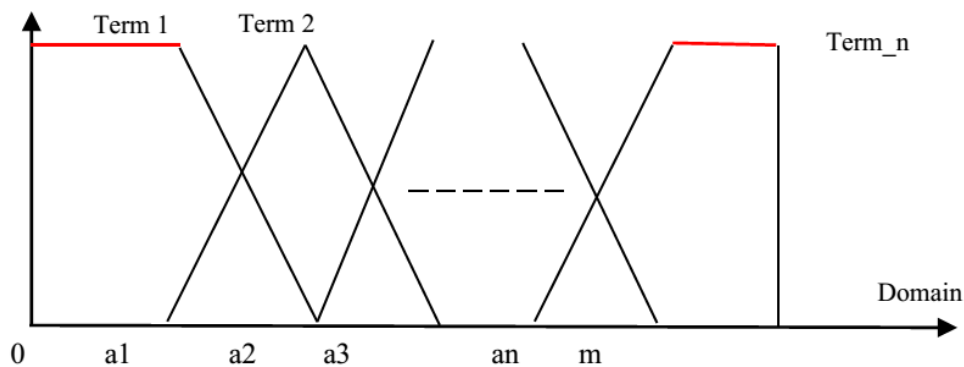


Fig.1 The Shape of the Membership Functions using Fuzzy Logic.

For each linguistic variable, first the number and the names of the terms of its domain must be declared, and after that the values of the parameters a_1, a_2, \dots, a_n must be specified.

a_1	a_2	a_3	a_n
Term 1	Term 2	Term 3	Term n

Domain1

III. THE GEOQUORUMS APPROACH

In this paper the Geoquorum algorithm to read/write in shared memory of mobile ad hoc networks is specified using fuzzy logic. In this paper the GeoQuorums approach has presented for implementing atomic read/write shared memory in 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 faulttolerant 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 [2]. 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 it 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 [3]. 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 associate s 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 t his 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 [3].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 putquorums 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 point 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 which is terminated rapidly in [3].

- 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 $OP = \{\text{get, put, confirm, recon- done}\}$.
- 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 .2 Notations Used in The Geoquorums Algorithm.

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 .3) [2].

Put/get variable type \mathcal{T}

State

Tag $\in T$, initially $\langle 0, i_0 \rangle$

Value $\in V$, initially v_0

Config-id $\in C$, initially $\langle 0, i_0, c_0 \rangle$

Confirmed-set $\subseteq T$, initially \emptyset

Recon-ip, a Boolean, initially false

Operations

Put (new-tag, new-value, new-Config-id)

If (new-tag > tag) then

Value \leftarrow new-value

Tag \leftarrow new-tag

If (new-Config-id > Config-id) then

Config-id \leftarrow new-config-id

Recon-ip \leftarrow true

Return put-Ack (Config-id, recon-ip)

Get (new-config-id)

If (new-config-id > Config-id) then

Config-id \leftarrow new-Config-id

Confirmed \leftarrow (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.3 Definition of the Put/Get Variable Type τ

A. 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 [4] .

1) **Operation Manager Client:** This automaton receives read, write, and recon requests from clients and manages quorum accesses to implement these operations (see fig. 4). 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 [3]:

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

 $\sqrt{C'} \in M, \exists P \in \text{put-quorums}(C'): P \subseteq \text{op. acc}$

Else

 $\exists P \in \text{put-quorums}(C): P \subseteq \text{Op. acc}$

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, \perp , \top , recon-ip, <0, i0, c0>, Ø >Output read-ack (v)_i

Precondition:

Conf-id=<time-stamp, Pid, c>

If op. recon-ip then

 $\sqrt{C'} \in M, \exists G \in \text{get-quorums}(C'): G \subseteq \text{op. acc}$

Else

 $\exists G \in \text{get-quorums}(C): G \subseteq \text{op. acc}$

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>
 $\sqrt{C'} \in M, \exists G \in \text{get-quorums}(C'): G \subseteq \text{op. acc}$
 Else
 $\exists G \in \text{get-quorums}(C): G \subseteq \text{op. acc}$
 Op. phase=get
 Op. type=read
 Op. tag \notin confirmed
 Effect:
 Current-port-number ←
 Current-port-number +1
 Op. phase ← put
 Op. Recon. ip ← recon-ip
 Op. acc ← \emptyset
 Output read-Ack (v)_i
 Precondition:
 Conf-id=<time-stamp, Pid, c>
 If op. recon-ip then
 $\sqrt{C'} \in M, \exists P \in \text{put-quorums}(C'): P \subseteq \text{op. acc}$
 Else
 $\exists P \in \text{put-quorums}(C): P \subseteq \text{op. acc}$
 Op. phase=put
 Op. type=read
 v=op. value
 Effect:
 Op. phase ← idle
 Confirmed ← confirmed-set 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, \perp , \perp , true, conf-id, \emptyset >
 Internal recon-2(cid)_i
 Precondition
 $\sqrt{C'} \in M, \exists G \in \text{get-quorums}(C'): G \subseteq \text{op. acc}$
 $\sqrt{C'} \in M, \exists P \in \text{put-quorums}(C'): P \subseteq \text{op. acc}$
 Op. type=recon
 Op. phase=get
 Cid=op. recon-conf-id
 Effect
 Current-port-number ←
 Current-port-number +1
 Op. phase ← put
 Op. acc ← \emptyset
 Output recon-Ack(c)_i
 Precondition

Effect:
Clock ← 1

Fig.4 Operation Manager Client Read/Write/Recon and Geo-update Transitions for Node

B. 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 [3]. 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. 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 [3]. 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 .5. 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 [4].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 [3].

Focal Point Emulator Server Transitions

Internal join ()_{obj, i}

Precondition:

Location \in FP-location

Status=idle

Effect:

Join-id \leftarrow <clock, i>Status \leftarrow joining

Enqueue (Lbcast-queue, <join-req, join-id>)

Input Lbcast-rcv (<join-req, jid>)_{obj, i}

Effect:

If ((status=joining) \wedge (jid=Join-id)) thenStatus \leftarrow listeningIf ((status=active)) \wedge jid \notin answered-join-reqs)) then

Enqueue (Lbcast-queue, <join-ack, jid, Val>)

Input Lbcast-rcv (<join-ack, jid, v>)_{obj, i}

Effect:

Answered-join-reqs \leftarrow answered-join-reqs \cup {jid}

```

If ((status=listening)  $\wedge$  (jid =join-id)) then
Status  $\leftarrow$  active
Val  $\leftarrow$  V
Input Geocast -rcv (< invoke, inv, oid, loc, FP-loc>)obj,i
Effect:
If (FP-loc=FP-location) then
If (<inv, oid, loc>  $\notin$  pending-ops U completed ops) then
Enqueue (Lbcst-queue, <invoke, inv, oid, loc>)
Input Lbcst -rcv (< invoke, inv, oid, loc>)obj,i
Effect:
If ((status=listening  $\vee$  active)  $\wedge$ 
(<inv, oid, loc>  $\notin$  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) $\leftarrow$   $\delta$  (inv, Val)
Enqueue (geocost- queue, < response, resp, oid, loc>)
Enqueue (completed-ops, Dequeue (pending-ops))
Internal leave ( )obj,i
Precondition:
Location  $\notin$  fp-location
Status  $\neq$  idle
Effect:
Status $\leftarrow$  idle
Join-id $\leftarrow$  <0, i0>
Val  $\leftarrow$  v0
Answered -join- reqs $\leftarrow$   $\emptyset$ 
Pending -ops  $\leftarrow$   $\emptyset$ 
Completed-ops  $\leftarrow$   $\emptyset$ 
Lbcst-queue  $\leftarrow$   $\emptyset$ 
Geocast-queue  $\leftarrow$   $\emptyset$ 
Output Lbcst (m)obj, i
Precondition:
Peek (Lbcst-queue) =m
Effect:
Dequeue (Lbcst- queue)
Output geocast (m)obj, j
Precondition:
Peek (geocast-queue) =m
Effect:
Dequeue (geocost- queue)
Input get-update (l, t)obj, i
Effect: Location  $\leftarrow$  l , Clock $\leftarrow$  t

```

Fig. 5 FPE Server Transitions for Client i and Object Obj of Variable Type $\mathcal{T} = \langle V, v_0, \text{invocations, responses, } \delta \rangle$

A) The Specification of the Geoquorum Approach Using Fuzzy Logic

A component repository contains several implementations of components that have the functionality of the application, specified with the provided property reading / writing in mobile ad hoc networks. Let us consider ed two different components, C1 and C2, specified as follows:

Component C1:

Property reading / writing with attributes

Read/ write _ACK_ rate = Crisp (0.2)

Read/ write _ACK_ rate = Crisp (0.4)

Occurrence = fuzzy (connect, about right, Almost no –connect)

Component C2:

Property reading / writing with attributes

Read/ write _ACK_ rate = Crisp (0.6)

Read/ write _ACK_ rate = Crisp (0.8)

Occurrence = fuzzy (connect, about right, Almost no connect)

Each of these attributes is defined as a linguistic variable with these terms as follows:

Domain (read/ write _ACK rate) = {ACK_ response, no change is needed, Almost no response}

Domain (occurrence) = {connect, about right, almost no connect}

For each linguistic variable, the parameters a_1 , a_2 , a_3 defining the shape of the membership functions are defined. In our application, in case of the attribute reading / writing, these values are ($a_1 = 0.2$), ($a_2 = 0.4$), ($a_3 = 0.6$), ($a_4 = 0.8$), and random values are ($a_5 = 0.1$), ($a_6 = 0.3$). It is important to note that a linguistic variable that characterizes an attribute can have different meanings in the context of different properties. The domain and the shape of a linguistic variable can be redefined in the context of different properties.

B) Generation of Fuzzy Rules

A new property matching mechanism is defined. In general, a requirement as: Requirement property P with attributes $A_1 = V_1$ and $A_2 = V_2$ and $A_n = V_n$ is handled in the following manner: First, the basic functionality is ensured, matching properties names according to the classical reading / writing strategy. Usually several solutions result from this first step. Second, the preliminary solutions are selected and hierarchies according to the degree of attribute matching [5] [6]. This is done by fuzzy logic. The given requirement is translated into the corresponding rule:

If $A_1 = V_1$ and $A_2 = V_2$ and ... $A_n = V_n$ then decision = select

The generation of the fuzzy rules is done automatically starting from the requirements. Very often, the required attributes are not values, but rather are required to be at least (or at most) a given value, $A < = V$ or $A > = V$. In general, a requirement containing the attribute expression $A < = V$ will be translated into a set of I rules, for all $V_i = < V$: If $A = V_i$ then decision = select

C) Extension of the Fuzzy Rules

Several rules are generated from one requirement. In order to relax the selection, it is considered a match even if one of the linguistic variables in the premises matches only a neighbor of the requested value (the predecessor or the successor). In this case the decision of selection is a weak one. In the case that more than one linguistic variable in the premise matches only neighbor values (while the rest match the requested fuzzy terms); the decision is a weak reject. In the extreme case that all linguistic variables in the premises match neighbor values, the decision is a weak reject [5] [7]. In all the other cases, the decision is a strong reject

For example, in the case of a requirement containing two attributes, $A_1 = V_1$ and $A_2 = V_2$, the complete set of generated rules is [7] [8]:

The directly generated rule is:

If $A_1 = V_1$ and $A_2 = V_2$ then decision = strong _ select

The rules generated if one of the linguistic variables in the premises matches only a neighbor of the requested value are (maximum 4 rules):

If $A_1 = \text{pred}(V_1)$ and $A_2 = V_2$ then decision = weak _ select

If $A_1 = \text{succ}(V_1)$ and $A_2 = V_2$ then decision = weak _ select

If $A_1 = V_1$ and $A_2 = \text{pred}(V_2)$ then decision = weak _ select

If $A_1 = V_1$ and $A_2 = \text{succ}(V_2)$ then decision = weak _ select

In this case there are a maximum number of four generated rules [9] [10]. If neither V_1 nor V_2 are extreme values of their domains, if a value is the first value in the domain it has no predecessor, if it is the last value in the domain it has no successor [11][12]. The rules generated if more than one of the linguistic variables in the premises matches only a neighbor of the requested value are (maximum 4 rules):

If $A1 = \text{pred}(V1)$ and $A2 = \text{pred}(V2)$ then decision = weak_reject
 If $A1 = \text{succ}(V1)$ and $A2 = \text{pred}(V2)$ then decision = weak_reject
 If $A1 = \text{pred}(V1)$ and $A2 = \text{succ}(V2)$ then decision = weak_reject
 If $A1 = \text{succ}(V1)$ and $A2 = \text{succ}(V2)$ then decision = weak_reject
 For all the rest of possible combinations of values of $A1$ and $A2$ the decision is strong-reject [13].

D) Specifying the Application using Fuzzy Rules

The rules generated for one different neighbor are [14] [15]:

[R1] If read/ write_Ack_rate = Almost_no response and occurrence = about right then decision = weak_select.

[R2] If read/ write_Ack_rate = Ack_response and occurrence = about right then decision = weak_select.

[R3] If read/ write_Ack_rate = Ack_response and occurrence = Almost no_connect then decision = weak_select.

[R4] If read/ write_Ack_rate = no change need and occurrence = connect then decision = weak_select

The rules generated for two different neighbors are:

[R5] If read/ write_Ack_rate = Almost_no response and occurrence = almost no_connect then decision = weak_reject.

[R6] If read/ write_Ack_rate = Ack_response and occurrence = almost no_connect then decision = weak_reject.

[R7] If read/ write_Ack_rate = Almost no response and occurrence = connect then decision = weak_reject.

[R8] If read/ write_Ack_rate = Ack_response and occurrence = connect then decision = weak_reject.

Fig 6-11 (a, b, c, d) illustrate how each of the generated rules is composed with the fact represented by the specification of component c_1 (with read/ write- Ack- rate= 0.1, 0.3, 0.2, 0.4, 0.8, 0.6 and occurrence= Almost no connect).

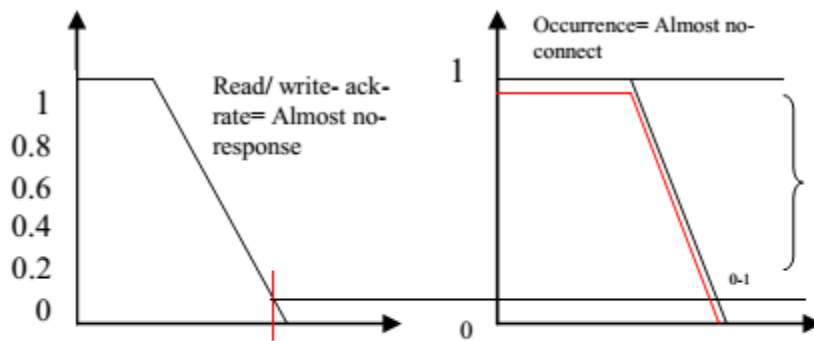


Fig (6-a): Rule: If read/ write- Ack- rate= (Almost- no- response) and occurrence= (almost- no- connect) then decision = weak-reject. Facts: read/ write- ask- rate= 0.1, occurrence= Almost no-connect.

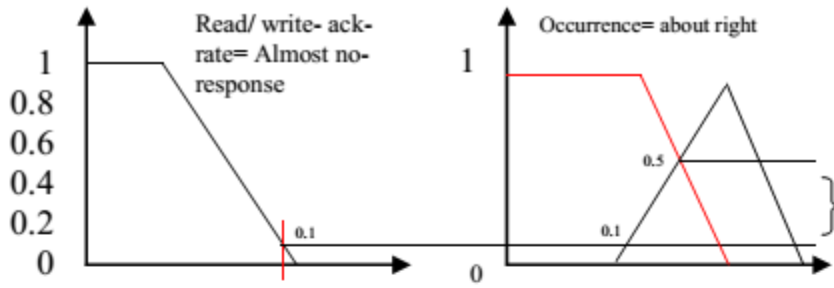


Fig (6-b): Rule: If read/ write- Ack- rate= Almost no- response and occurrence= about right then decision= weak- reject. Facts: read/ write- Ack- rate= 0.1 occurrence= Almost no- connect.

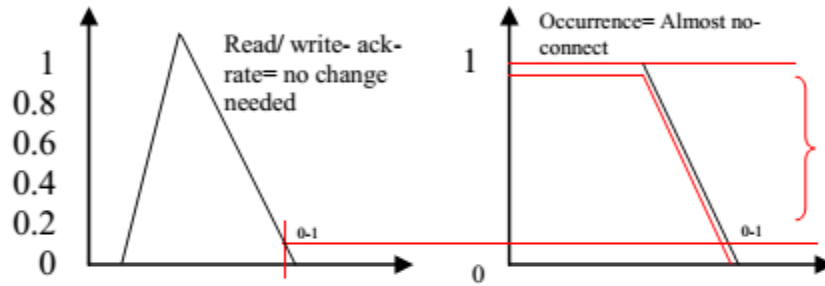


Fig (6-C): Rule: If read/ write- Ack- rate= no change needed and occurrence= Almost no- connect then decision= weak- reject facts: read/ write- ack- rate= 0.1, (occurrence= Almost no- connect)

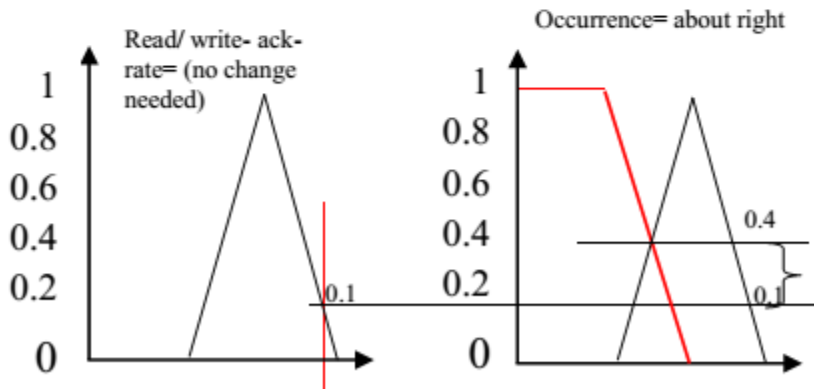


Fig (6-d): Rule: If read/ write- Ack – rate= no change needed and occurrence= about right then decision= weak – reject. Facts: read/ write- Ack- rate= 0.1, occurrence= almost no- connect

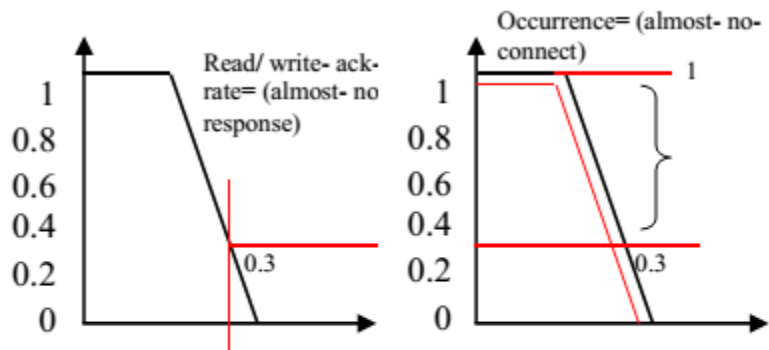


Fig (7-a): Rule: If read/ write- Ack – rate= (Almost no- response) and occurrence= Almost no- connect then decision= weak – reject
 Facts: read/ write- Ack- rate= 0.3, occurrence= almost no- connect

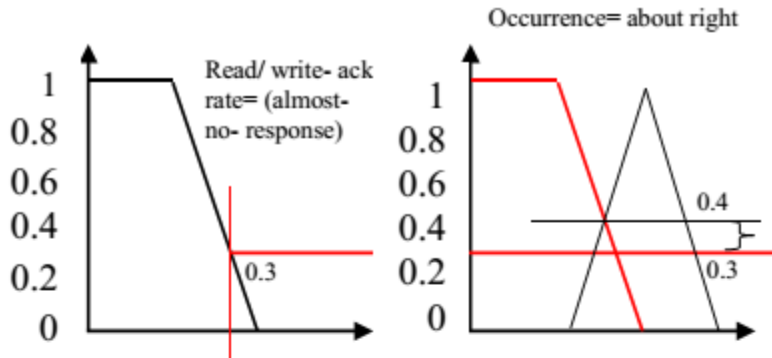


Fig (7-b): Rule: If read/ write- Ack – rate= (Almost no- response) and occurrence= about right then decision= strong – select. Facts: read/ write- Ack- rate= 0.3, occurrence= almost no- connect

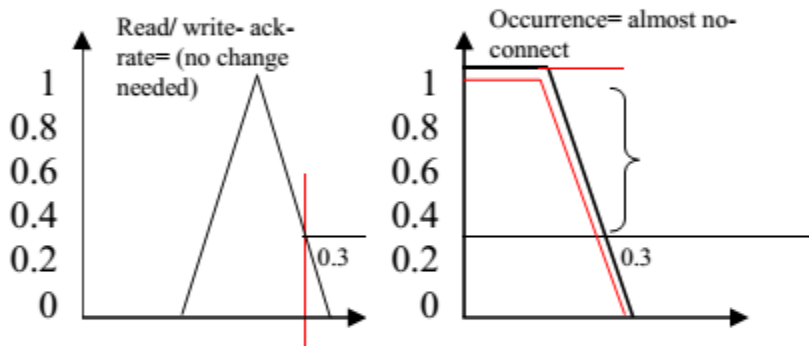


Fig (7-c): If read/ write- Ack – rate= no change needed and occurrence= Almost no- connect then decision= weak–reject Facts: read/ write- Ack- rate= 0.3, occurrence= almost no- connect

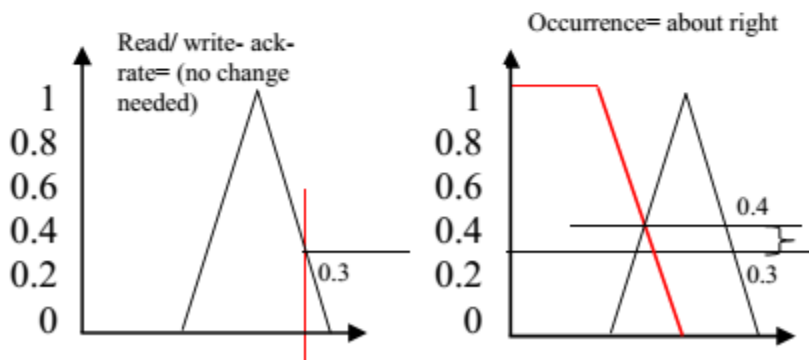


Fig (7-d): Rule: If read/ write- Ack – rate = no change needed and occurrence= about right then decision= strong- select .Facts: read/ write- Ack- rate= 0.3, occurrence= almost no- connect

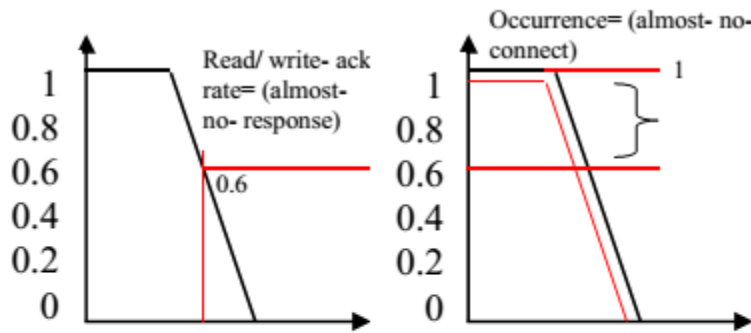


Fig (8-a): Rule: If read/ write- Ack – rate= (Almost no- response) and occurrence= Almost no- connect then decision= weak – reject
 Facts: read/ write- Ack- rate= 0.6, occurrence= almost no- connect.

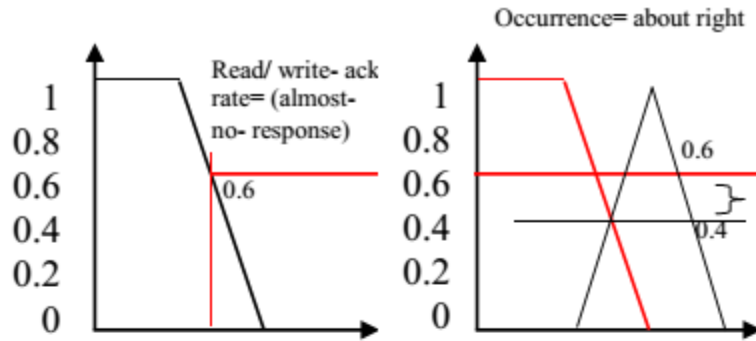


Fig (8-b): Rule: If read/ write- Ack – rate= (Almost no- response) and occurrence= about right then decision= weak – reject. Facts: read/ write- Ack- rate= 0.6, occurrence= almost no- connect

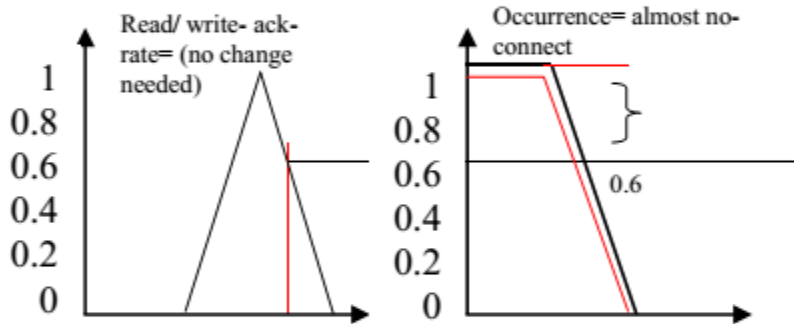


Fig (8-c): Rule: If read/ write- Ack – rate= no change needed and occurrence= Almost no- connect then decision= weak – reject facts: read/ write- Ack- rate= 0.6, occurrence= almost no- connect

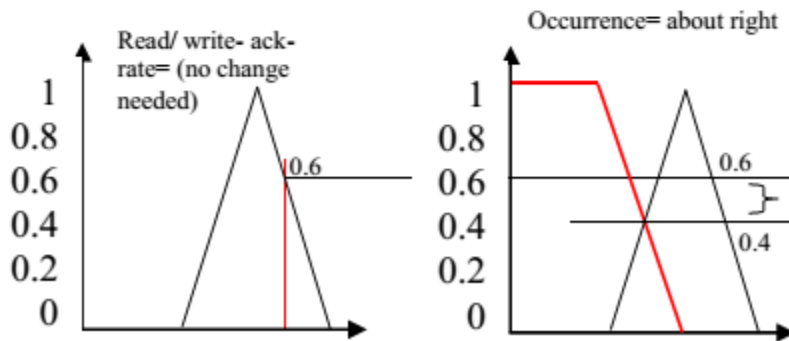


Fig (8-d): Rule: If read/ write- Ack – rate= no change needed and occurrence= about right then decision= weak – select. Facts: read/ write- Ack- rate= 0.6, occurrence= almost no- connect

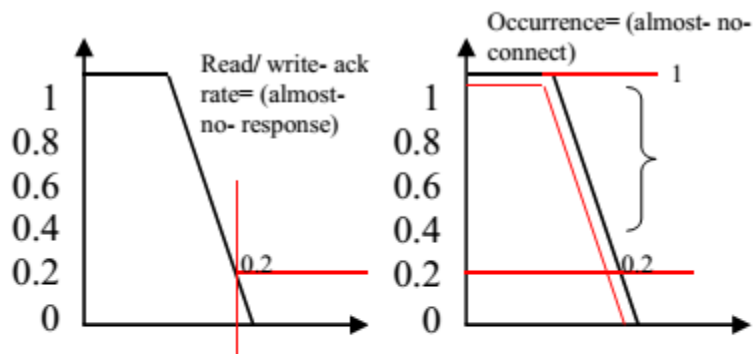


Fig (9-a): Rule: If read/ write- Ack – rate= (Almost no- response) and occurrence= almost no- connect then decision= weak – reject
 Facts: read/ write- Ack- rate= 0.2, occurrence= almost no- connect

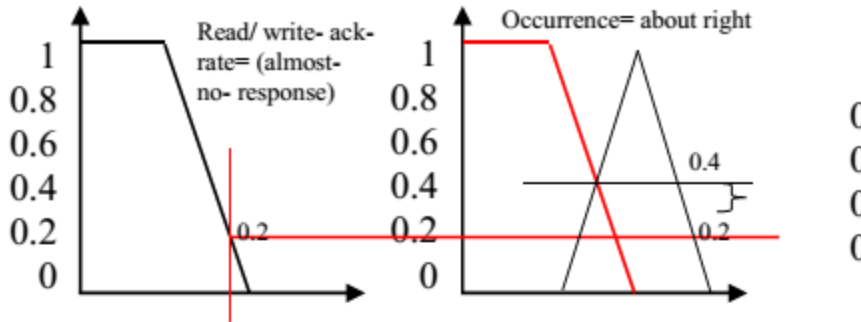


Fig (9-b): Rule: If read/ write- Ack – rate= (Almost no- response) and occurance= about right then decision= weak – select. Facts: read/ write- Ack- rate= 0.2, occurance= almost no- connect

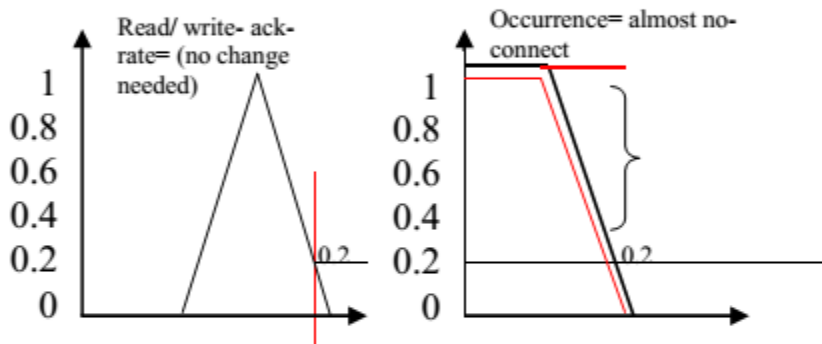


Fig (9-c): Rule: If read/ write- Ack – rate= no change needed and occurance= Almost no- connect then decision= weak – reject. Facts: read/ write- Ack- rate= 0.2, occurance= almost no- connect

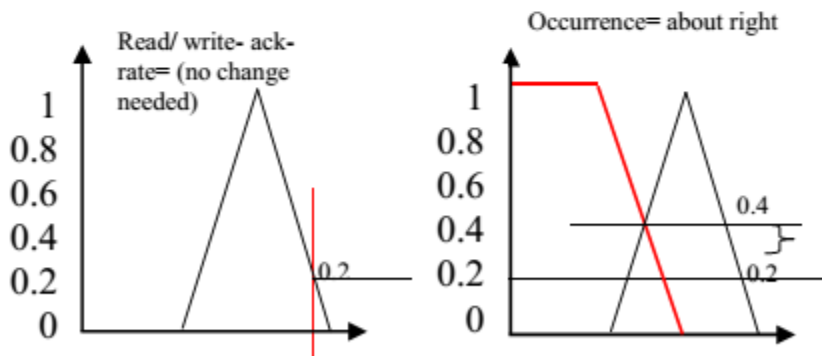


Fig (9-d): Rule: If read/ write- Ack – rate= no change needed and occurance= about right then decision= weak – select. Facts: read/ write- Ack- rate= 0.2, occurance= almost no- connect

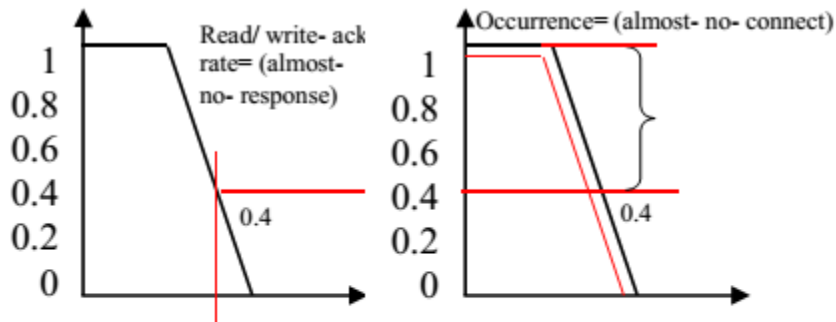


Fig (10-a): Rule: If read/ write- Ack - rate= (Almost no-response) and occurrence= Almost no- connect then decision= weak - reject facts: read/ write- Ack- rate= 0.4, occurrence= almost no- connect

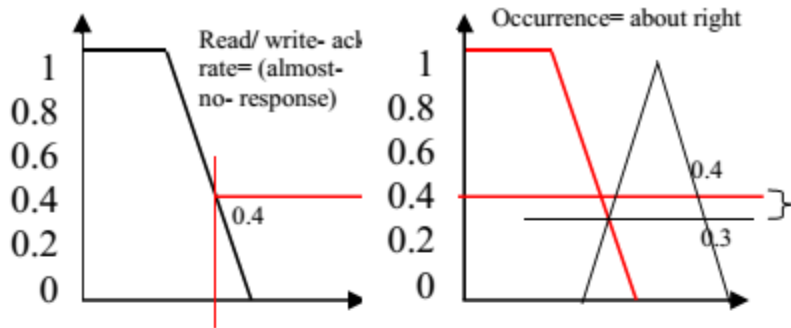


Fig (10-b): Rule: If read/ write- Ack – rate= (Almost no-response) and occurrence= about right then decision= strong-select. Facts: read/ write- Ack- rate= 0.4, occurrence= almost no-connect

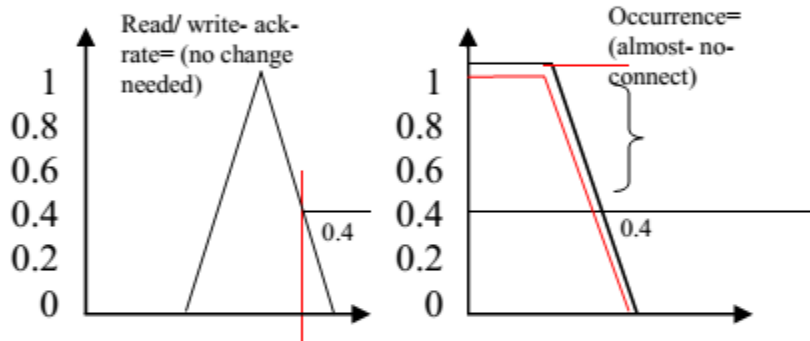


Fig (10-c): Rule: If read/ write- Ack – rate= no change needed and occurrence= Almost no- connect then decision= weak – reject. Facts: read/ write- Ack- rate= 0.4, occurrence= almost no- connect

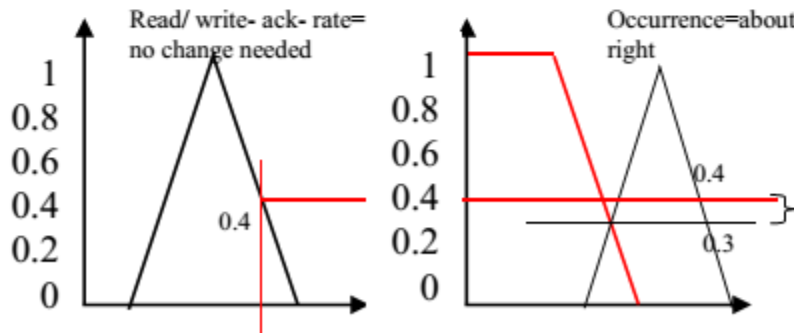


Fig (10-d): Rule: If read/ write- Ack – rate= no change needed and occurrence about right then decision= strong-select. Facts: read/ write- Ack- rate= 0.4, occurrence= almost no- connect

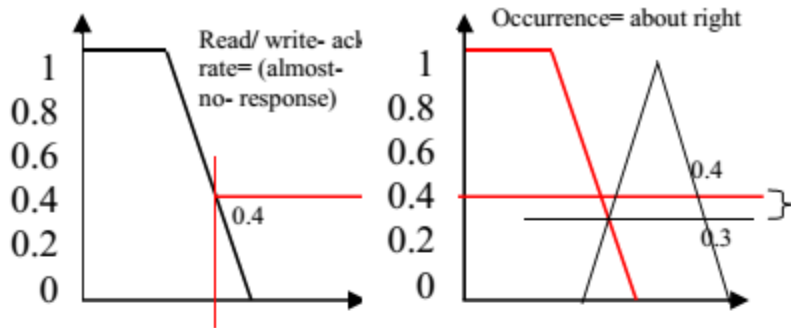


Fig (10-b): Rule: If read/ write- Ack – rate= (Almost no-response) and occurrence= about right then decision= strong-select. Facts: read/ write- Ack- rate= 0.4, occurrence= almost no-connect

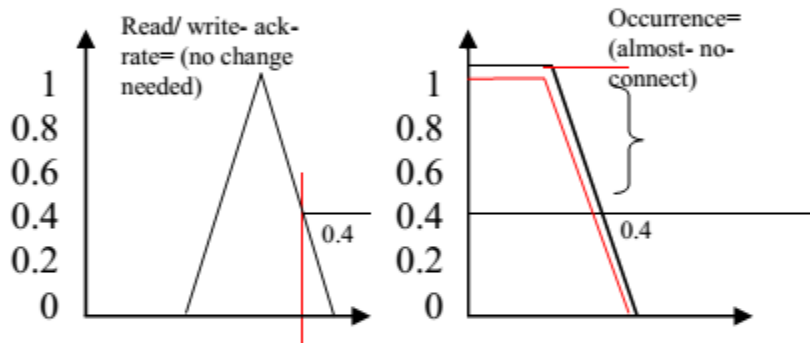


Fig (10-c): Rule: If read/ write- Ack – rate= no change needed and occurrence= Almost no- connect then decision= weak – reject. Facts: read/ write- Ack- rate= 0.4, occurrence= almost no- connect

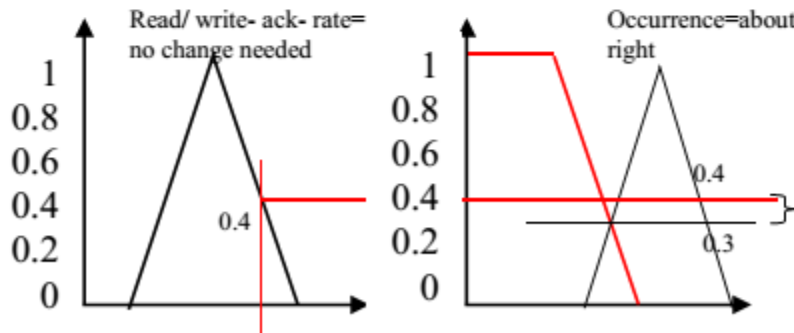


Fig (10-d): Rule: If read/ write- Ack – rate= no change needed and occurrence about right then decision= strong-select. Facts: read/ write- Ack- rate= 0.4, occurrence= almost no- connect

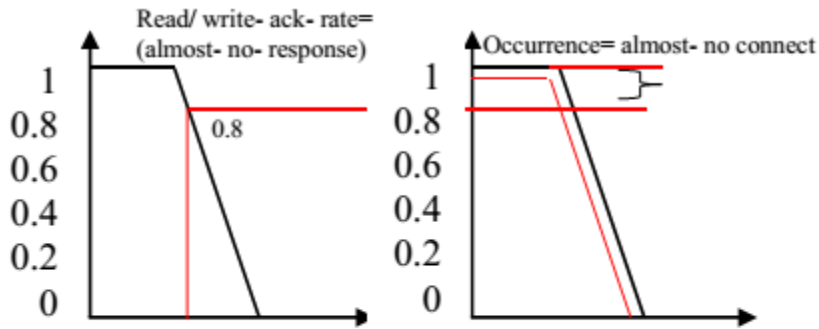
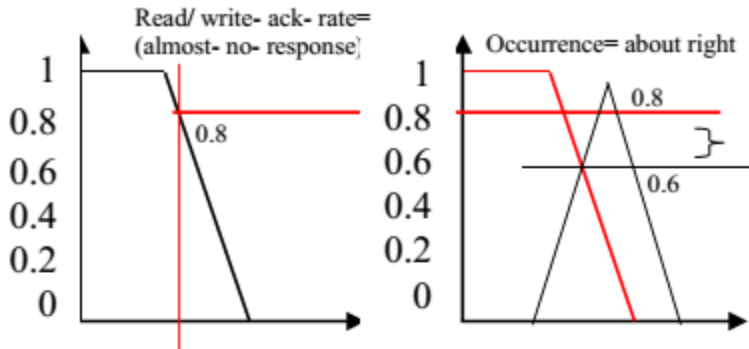


Fig (11-a): Rule: If read/ write- Ack - rate= (Almost no-response) and occurrence= Almost no- connect then decision= weak - select Facts: read/ write- Ack- rate= 0.8, occurrence= almost no- connect



(11-b): Rule: If read/ write- Ack - rate= (Almost no-response) and occurrence= about right then decision= weak - select. Facts: read/ write- Ack- rate= 0.8, occurrence= almost no-connect

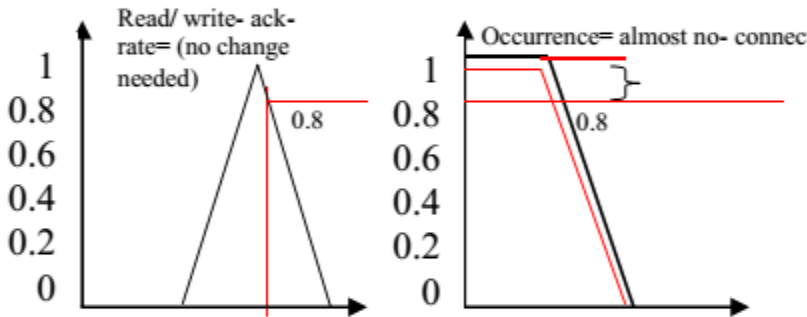


Fig (11-c): Rule: If read/ write- Ack - rate= no change needed and occurrence= Almost no- connect then decision= weak - select Facts: read/ write- Ack- rate= 0.8, occurrence= almost no- connect

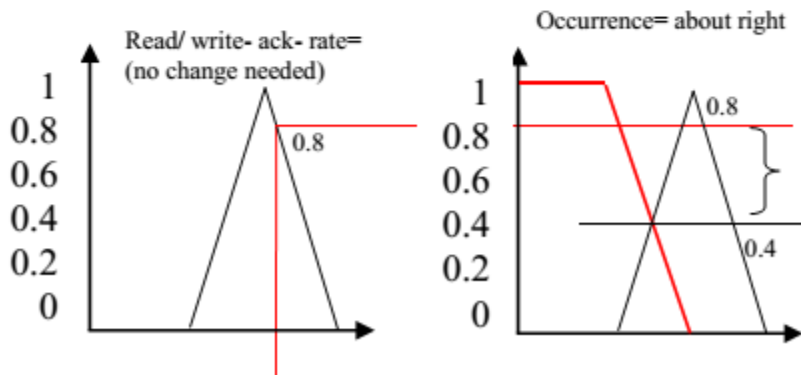


Fig (11-d): Rule: If read/ write- Ack - rate= no change needed and occurrence= about right then decision= weak - reject. Facts: read/ write- Ack- rate= 0.8, occurrence= almost no- connect

*E) Performance Evaluation for Geoquorum Approach:
Implementing Atomic Read/ Write Shared memory in
Mobile Ad hoc network using fuzzy logic.*

Let us consider these assumptions:

1-Input status word descriptions

Almost no- connect

About right

Connect

2- Output action word descriptions

Ack- response

No change needed

Almost no- response

3- Rules

Translate the above into plain English rules (Called linguistic Rules). These rules will appear as follow:

Rule 1: If the status is connect then Ack – response.

Rule 2: If the status is about right, then no change need

Rule 3: If the status is almost no- connect then Almost no- response.

4- The next (3 steps) use a charting technique, one function of the charting technique is to determine “The degree of membership” of: Almost no- connect, about right and connect triangles for a given values (see fig.12).

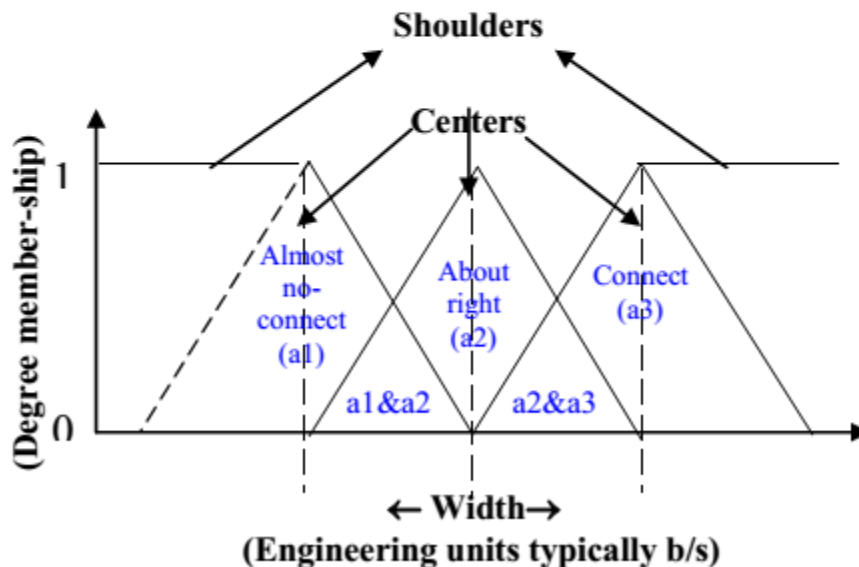
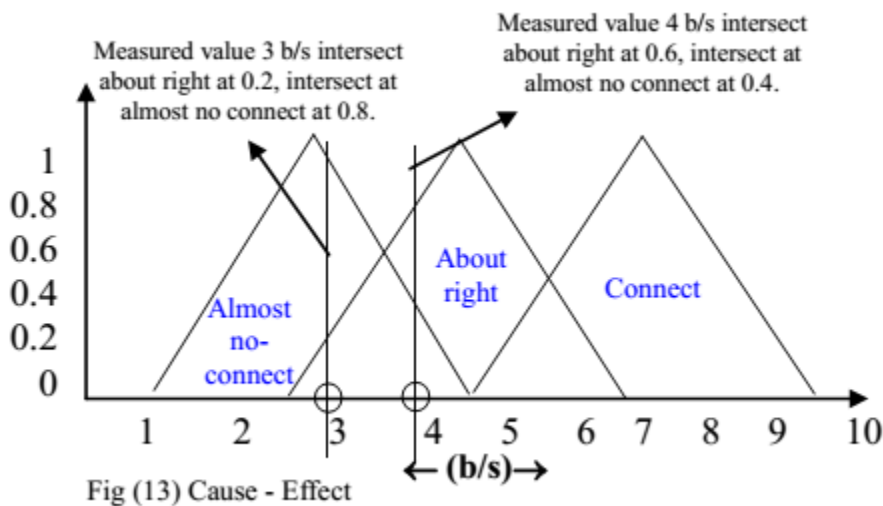


Fig (12) Membership Functions

5- Associate the above inputs and outputs as causes and

effects with a rules charts, as in the next fig.13 below, the chart is made with triangles, the use of which will be explained. Triangles work just fine and are easy to work with width of the triangles can vary [16] [17]. Narrow triangles provide tight control when operating conditions are in the area. Wide triangles provide looser control. Narrow triangles are usually used in the center, at the set point (the target value).



6- We draw “effect” (output determining) triangles with their value ($h=3$ b/s or 4 b/s and their multiplications) is determined. The triangles are drawn by the previous rules. Since the height doesn’t intersect with connect, so we don’t draw it in the following (Figure 13- (a) (b)). These “effect” triangles will be used to determine the controller output.

The result is affected by the width we have given the triangles and will be calculated. See fig 10 below the no change need has a height of 0.2,

0.6 and the Almost no- response has a height of 0.8, 0.4 because these were the intersect points for their matching “cause” triangles.

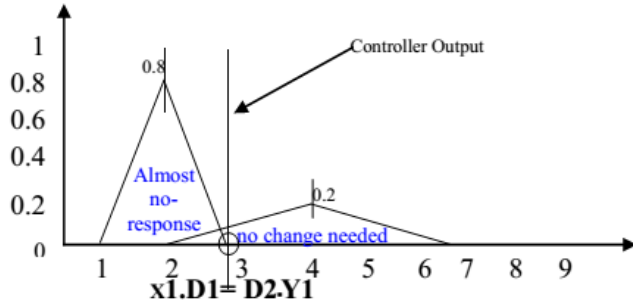


Fig. (13-a): Determination of controller output.

The output as seen in fig (10-a) is determined by calculating the point at which balance the two triangles, as follow:

The area of no change need triangle is $\frac{1}{2} \times 0.2 \times 5=0.5$

The area of Almost No- response triangle is $\frac{1}{2} \times 0.8 \times 2=0.8$

We are looking for the balance point; find the balance point with the following calculation:

Equation1: $0.8 \times D_1 = 0.5 \times D_2$

(D_1 is the fulcrum distance form X_1 and, D_2 is the fulcrum distance from Y_1)

Equation2: $D_1 + D_2 = 2.5 \rightarrow D_1 = 2.5- D_2$

So, by substituting $(2.5- D_2)$ for D_1 in equation1 gives D_2

$0.8 \times (2.5- D_2) = 0.5 D_2$

$2-0.8 D_2 = 0.5 D_2$

$2=1.3 D_2 \rightarrow D_2=1.5$

$\therefore D_1=1$

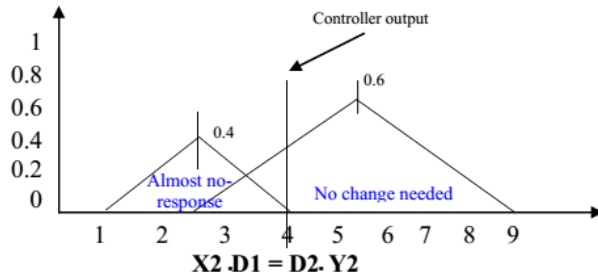


Fig (13-b): Determination of controller output.

The output as seen in fig (10-b) is determined by calculating the point at which balance the two triangles, as follow:

The area of no change need triangle is $\frac{1}{2} \times 6 \times 0.6= 1.8$

The area of Almost No- response triangle is $\frac{1}{2} \times 3 \times 0.4=0.6$

We are looking for the balance point; find the balance point with the following calculation:

Equation1: $0.6 \times D_1 = 1.8 \times D_2$

(D_1 is the fulcrum distance form X_2 and, D_2 is the fulcrum distance from y_2)

Equation2: $D_1 + D_2 = 3.5 \rightarrow D_1 = 3.5- D_2$

So, by substituting $(3.5- D_2)$ for D_1 in equation1 gives D_2

$0.6 \times (3.5- D_2) = 1.8 D_2$

$2.1- 0.6 D_2 = 1.8 D_2$

$2.1=1.8 D_2 + 0.6 D_2$

$2.1=2.4 D_2 \rightarrow D_2 \approx 0.9$

$\therefore D_1=2.6$

Note that, samples at instant times with a resulting controller output are discussing; the controller is sampling several times each second with a resulting “correction” output following each sample.

IV. CONCLUSIONS

In this paper a specification and performance evaluation for the Geoquorums approach for implementing atomic read/write shared memory in mobile ad hoc networks which is based on fuzzy logic is presented . The advantages of this solution are: a natural treatment for certain non-functional attributes that cannot be exactly evaluated and specified, and a relaxed matching of required/provided attributes that do not have to always be precise and consistent.

REFERENCES

- [1] Felix Bachman, Len Bass, C Buhman, S Comella-Dorda, F Long, J Robert, R Seacord, Kurt Wallnau," Technical concepts of componentbased software engineering," Technical Report CMU/SEI-2000-TR-008, Carnegie Mellon Software Engineering Institute, 2000**
- [2] Kendra Cooper, Joao Cangusu, Rong Lin, Ganesan Sankaranarayanan, Ragouramane Soundararadjane, Eric Wong," An Empirical Study on the Specification and Selection of Components Using Fuzzy Logic," in Proceedings of 8th International Symposium on CBSE, St. Louis, USA, May 2005.**
- [3] Dolv, S., Gilbert, S.Lynch, N.A., Shvartsman, A.A., Welch, A.Loran.J.L:"Geoquorums: Implementing Atomic Memory in Mobile Ad Hoc Networks ".In: Proceedings of the 17th International Conference on The Distributed Computing, PP: 306-319 (2005).**
- [4] Haas, Z.J., Liang, and B.A, D.Wjghs. "Ad Hoc Mobile Management with Uniform GeoQuorums Systems", In: Proceeding of IEEE/ACMTransactions on Mobile ad hoc Networks 7(2), PP: 228-240(2004).**
- [5] Murat Koyuncu, Adnan Yazici," A Fuzzy Knowledge-Based System for Intelligent Retrieval," in IEEE Transactions on Fuzzy Systems, Vol. 13, No. 3, June 2005, pp. 317-330**
- [6] Ioana Sora, Pierre Verbaeten, Yolande Berbers," A Description Language for Composable Components, in Fundamental Approaches to**

Software Engineering", Lecture Notes in Computer Science LNCS No. 2621, Springer, 2003, pp. 22-37

[7] Ioana Sora, Vladimir Cretu, Pierre Verbaeten, Yolande Berbers, "Automating decisions in component composition based on propagation of Requirements, in Fundamental Approaches to Software Engineering," Lecture Notes in Computer Science LNCS No. 2984, Springer, 2004, pp. 374-388

[8] Ioana Sora, Vladimir Cretu, Pierre Verbaeten, Yolande Berbers," Managing Variability of Self-customizable Systems through Composable Components, in Software Process Improvement and Practice, " Vol. 10, No. 1, Addison Wesley, January 2015

[9] Clemens Szyperski: Component Software: Beyond Object Oriented Programming, Addison Wesley, 2016

[10] Ting Zhang, Luca Benini, Giovanni De Micheli, "Component Selection and Matching for IP-Based Design," in Proceedings of Conference on Design, Automation and Test in Europe (DATE), Munich, Germany, 2017, pp. 40- 46.

[11] Ioana Şora, D. Todinca," Specification-based Retrieval of Software Components through Fuzzy Inference, in Acta Polytechnica Hungarica. Vol. 3, No. 3, 2016..

[12] R. Oliveira, L. Bernardo, and P. Pinto, "Modeling delay on IEEE 802.11 MAC protocol for unicast and broadcast non saturated traffic," in Proc. WCNC'07, IEEE, 2017, pp. 463–467.

[13] A. Fehnker, M. Fruth, and A. McIver, "Graphical modeling for simulation and formal analysis of wireless network protocols,"in Methods, Models and Tools for Fault Tolerance, LNCS 5454. Springer, 2019, pp. 1–24.

[14] F. Ghassemi, W. Fokkink, and A. Movaghar, "Restricted broadcast process theory," in Proc. SEFM'08, IEEE, 2018, pp. 345–354.

[15] F. Ghassemi, W. Fokkink, and A. Movaghar, “Equational reasoning on ad hoc networks,” in Proc. FSEN’09, LNCS 5961. Springer, 2018, pp. 113–128.

[16] T. Lin, “Mobile Ad-hoc Network Routing Protocols: Methodologies and Applications,” PhD thesis, Virginia Polytechnic Institute and State University, 2017.

[17] V. D. Tracy Camp, Jeff Boleng, “A survey of mobility models for ad hoc network research,” Wireless Communications and Mobile Computing, 2:483–502, 2018.