International Journal of Computer Applications (0975 – 8887)
Volume 27– No.4, August 201

Constructing Java Code for Specification – based Approach for Implementing Atomic Read/ Write Shared Memory in Mobile Ad Hoc Networks Using Fuzzy Logic

Sherif El-Etriby
Faculty of Computers and Information,
Computer Science Dept., Menoufiya University, Egypt.

Reham Shihata
Faculty of Science, Math's. Dept.,
Menoufiya University, Egypt.

ABSTRACT

In this paper we construct java code for specification phase of the Geoquorum approach: implementing atomic read/ write shared memory in mobile ad hoc networks using fuzzy logic. This code is based on graphical user interface and is considered a tool to determine the specification of the communication protocol based on asynchronous real time distributed system using java language. This code satisfies the accuracy and determines the requirements of the specification phase using fuzzy logic of our suggested application.

General Terms

Software Lifecycle, Wireless Network, Java Language.

Keywords

Specification Phase, Mobile Ad Hoc Network, Fuzzy Logic, Java.

1. 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 [1] [11]. 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 [11]. 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 presented on the flow connected to the logical point; these requirements are able to propagate. A property consists of a name describing functionality and attributes that are either type values or fuzzy terms. The names used for the properties and for the attributes are established through a domain-specific vocabulary[2][11] .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][11]. 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 shown in Fig.1.

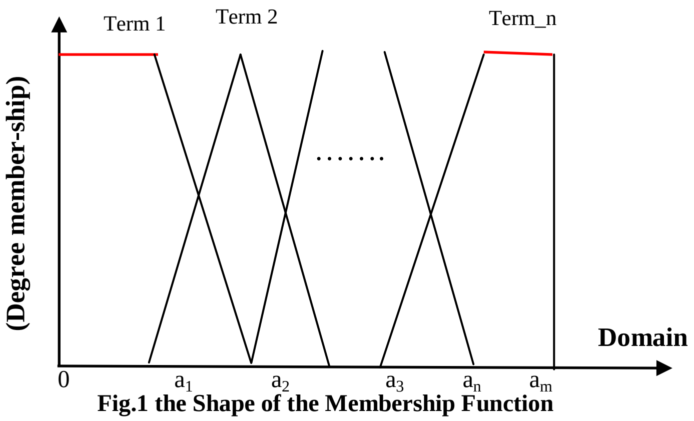


Fig.1 the Shape of the Membership Function

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… a_n must be specified. Also, we should define a java as a tool which is used in the specification phase of our suggested application. On a timesharing system, users sit at "terminals" where they type commands to the computer, and the computer types back its response. Early personal computers also used typed commands and responses, except that there was only one person involved at a time. This type of interaction between a user and a computer is called a command-line interface [3]. Today, of course, most people interact with computers in a completely different way. They use a Graphical User Interface, or GUI. The computer draws interface components on the screen. The components include things like windows, scroll bars, menus, buttons, and icons. Usually, a mouse is used to manipulate such components. Assuming that you have not just been teleported in from the 1970s, you are no doubt already familiar with the basics of graphical user interfaces. Computer users today expect to interact with their computers using a graphical user interface (GUI). Java can be used to write GUI programs ranging from simple applets which run on a Web page to sophisticated stand-alone applications [3]. GUI programs differ from traditional "straight-through" programs. One big difference is that GUI programs are event-driven. That is, user actions such as clicking on a button or pressing a key on the keyboard generate events, and the program must respond to these events as they occur. And of course, objects are everywhere in GUI programming. Events are objects. Colors and fonts are objects. GUI components such as buttons and menus are objects. Events are handled by instance methods contained in objects. In Java, GUI programming is object-oriented programming. A lot of GUI interface components have become fairly standard. That is, they have similar appearance and behavior on many different computer platforms including Macintosh, Windows, and Linux. Java programs, which are supposed to run on many different platforms without modification to the program, can use all the standard GUI components. They might vary a little in appearance from platform to platform, but their functionality should be identical on any computer on which the program runs. Now, Java actually has two complete sets of GUI components. One of these, the AWT or Abstract Windowing Toolkit, was available in the original version of Java. The other, which is known as Swing, is included in Java version 1.2 or later, and is used in preference to the AWT in most modern Java programs [4]. Also, try to get some feel about how object-oriented programming and inheritance are used here. Note that all the GUI classes are subclasses, directly or indirectly, of a class called *JComponent*, which represents general properties that are shared by all Swing components. Two of the direct subclasses of *JComponent* themselves have subclasses. The classes *JTextArea* and *JTextField*, which have certain behaviors in common, are grouped together as subclasses of *JTextComponent*. Similarly *JButton* and *JToggleButton* are subclasses of *JAbstractButton*, which represents properties common to both buttons and checkboxes. (*JComboBox*, by the way, is the Swing class that represents pop-up menus.)[4]. Just from this brief discussion, perhaps you can see how GUI programming can make effective use of object-oriented design. In fact, GUI's, with their "visible objects," are probably a major factor contributing to the popularity of OOP.

2. ATOMIC READ/WRITE SHARED MEMORY IN MOBILE AD HOC NETWORK

In this paper the Geoquorum 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 [5]. The Geoquorum algorithm uses the fault-prone focal point objects to implement atomic read/write operations on a fault-tolerant virtual shared object. The Geoquorum 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. Overall, the new Geoquorum algorithm efficiently implements read/write operations in a highly dynamic, mobile network. In this study the basic idea for the proposed approach is 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 [5]. 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 [5] [6]. 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 explained as 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 [5] [6]. 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. 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 [5]. (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 Geoquorum 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 [5] [6]. 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 [5][6]. 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. 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 see the following illustration table; Tab 1. Notations Used in the Geoquorum Algorithm. A read/write object has the following variable type as it can be seen in Tab.2 in Put/Get Variable type τ .

Tab. 1: Notations Used in the Geoquorum Algorithm.

I	The totally- ordered set of node identifiers.
I_0 є I	A distinguished node identifier in I that is smaller than all order identifiers in I.
S	The set of port identifiers, defined as $N^{>0}$× OP×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 $R^{\geq 0}$ × I.
U	The set of operation identifiers, defined as $R^{\geq 0}$ × S.
X	The set of memory locations for each x є X: V_x the set of values for x , $v_{0,x}$ є V_x , the initial value of X.
M	A totally-ordered set of configuration names
C_0 є 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: $R^{\geq 0}$ ×I ×M
L	Set of locations in the plane, defined as R× R

Tab.2: Put/Get Variable type τ

STATES	Config-id ∈ C, initially< 0, i_0, c_0>
	Confirmed-set C T, initially Ø
	Recon-ip, a Boolean, initially false
OPERATIONS	Put (new-tag, new-value, new-Config-id)
	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
	Confirmed ← (tag ∈ 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 ( )

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. This automaton receives read, write, and recon requests from clients and manages quorum accesses to implement these operations (see fig. 2). 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 [5] [6]:

```
Operation Manager Client Transitions
Input write (Val) $_i$
Current-port-number $\longleftarrow$
Current-port-number +1
Op $\longleftarrow$ < 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
√ $C' \in$ M, э P $\in$ put-quorums($C'$): P $\underline{C}$ op. acc
Else
Э P $\in$ put-quorums(C): P $\underline{C}$ Op. acc
Op .phase=put
Op. type=write
Op. phase $\longleftarrow$ idle
Confirmed $\longleftarrow$ confirmed U {op. tag}

Input read ( )$_i$
Current-port-number $\longleftarrow$

Current-port-number +1
Op $\longleftarrow$ < 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
√ $C' \in$ M, э G $\in$ get-quorums($C'$): G $\underline{C}$ op. acc
Э G $\in$ get-quorums(C): G $\underline{C}$ op. acc
Op. phase=get
Op. type=read
Op. tag $\in$ confirmed
v= op. value
Op .phase $\longleftarrow$ idle
Internal read-2( )$_i$
Precondition:
Conf-id=<time-stamp, Pid, c>
√ $C' \in$ M, э G $\in$ get-quorums($C'$): G $\underline{C}$ op. acc
Э G $\in$ get-quorums(C): G $\underline{C}$ op. acc
Op. phase=get
Op. type=read
Op. tag $\notin$ confirmed
Current-port-number $\longleftarrow$
Current-port-number +1
Op. phase $\longleftarrow$ put
Op. Recon. ip $\longleftarrow$ recon-ip
Op. acc $\longleftarrow$ Ø
Output read-Ack (v)$_i$
Precondition:
Conf-id=<time-stamp, Pid, c>
If op. recon-ip then
√ $C' \in$ M, э P $\in$ put-quorums($C'$): P $\underline{C}$ op. acc
Э P $\in$ put-quorums(C): P $\underline{C}$ op. acc
Op. phase=put
Op. type=read
v=op. value
Op. phase $\longleftarrow$ idle
Confirmed $\longleftarrow$ confirmed-set U {op. tag}
Input recon (conf-name)$_i$
Conf-id $\longleftarrow$ <clock, i, conf-name>
Recon-ip $\longleftarrow$ true
Current-port-number $\longleftarrow$
Current-port-number +1
Op $\longleftarrow$ < recon, get, $\perp$, $\perp$, true, conf-id, Ø>
Internal recon-2(cid)$_i$
Precondition
√ $C' \in$ M, э G $\in$ get-quorums($C'$): G $\underline{C}$ op. acc
√ $C' \in$ M, э P $\in$ put-quorums($C'$): P $\underline{C}$ op. acc
Op. type=recon
Op. phase=get
Cid=op. recon-conf-id
Current-port-number $\longleftarrow$
Current-port-number +1
Op. phase $\longleftarrow$ put
Op. acc $\longleftarrow$ Ø
Output recon-Ack(c) $_i$
Precondition
Cid=op. recon-conf-id
Cid= <time-stamp, Pid, c>
Э P $\in$ put-quorums(C): P $\underline{C}$ op. acc
Op. type=recon
Op. phase=put
If (conf-id=op. recon-conf-id) then
Recon-ip $\longleftarrow$ false
Op. phase $\longleftarrow$ idle
Input geo-update (t, L) $_i$
Clock $\longleftarrow$ 1
```

Fig .2 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 [5]. 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 [6]. 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. 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[5][6]. 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 are completed 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 has been broadcast but before it receives 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 receives the snapshot and has the active status. A precondition for performing most of these actions happens when 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 [5][6].

3. 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 considered 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 status) = {ACK_ response, no change is needed, Almost no response}

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

For each linguistic variable set of 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.

3.1 Generation of Fuzzy Rules

A new property matching mechanism is defined. In general, a requirement as: Requirement property P with attributes A1 = V1 and A2 = V2 and An = Vn 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 [7] [8] [11]. This is done by fuzzy logic. The given requirement is translated into the corresponding rule:

If A1= V1 and A2 = V2 and … An = Vn 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 Vi >= V: If A= Vi then decision = select

3.2 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) [7] [8] [11]. 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 [7]. In all the other cases, the decision is a strong reject. For example, in the case of a requirement containing two attributes, A1= V1 and A2=V2, the complete set of generated rules is [7] [8] [11]:

The directly generated rule is:

If A1=V1 and A2=V2 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) [7] [8] [11]:

If A1 = pred (V1) and A2=V2 then decision = weak _ select

If A1 = succ (V1) and A2= V2 then decision =weak _select

If A1 = V1 and A2 = pred (V2) then decision = weak _ select

If A1 = V1 and A2 = succ (V2) then decision =weak _select

In this case there are a maximum number of four generated rules [9] [10]. If neither V1 nor V2 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] [13]. 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= pred (V1) and A2 = pred (V2) then decision =weak_ reject

If A1= succ (V1) and A2 = pred (V2) then decision =weak_ reject

If A1= pred (V1) and A2 = succ (V2) then decision =weak_ reject

If A1= succ (V1) and A2 = succ (V2) then decision =weak_ reject

For all the rest of possible combinations of values of A1and A2 the decision is strong-reject [11].

3.3 Specifying the Application using Fuzzy Rules

The rules generated for one different neighbor are:

[R1] If read/ write_ Ack_ status = Almost_ no response and occurrence = about right then decision = weak_ select.

[R2] If read/ write_ Ack_ status = Ack_ response and occurrence = about right then decision = weak_ select.

[R3] If read/ write_ Ack_ status = Ack_ response and occurrence = Almost no_ connect then decision = weak_ select.

[R4] If read/ write_ Ack_ status = no change need and occurrence = connect then decision = weak_ select

The rules generated for two different neighbors are:

[R5**]** If read/ write_ Ack_ status =Almost_ no response and occurrence = almost no- connect then decision = weak_ reject.

[R6] If read/ write_ Ack_ status = Ack_ response and occurrence = almost no_ connect then decision = weak_ reject.

[R7] If read/ write_ Ack_ status = Almost no response and occurrence = connect then decision =weak_ reject.

[R8] If read/ write_ Ack_ status = Ack_ response and occurrence = connect then decision = strong _ select. Fig 3 -8(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).

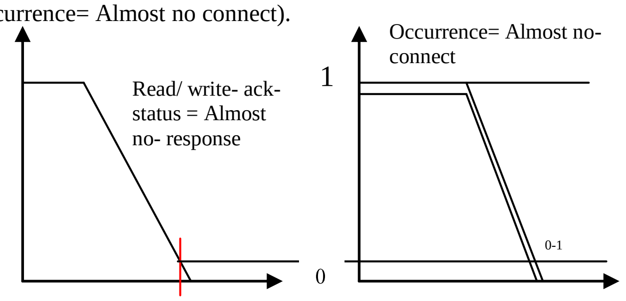

Read/ write- ack- status = Almost no- response
Occurrence= Almost no- connect
1
0
0-1

Fig (3-a): Rule: If read/ write- Ack- status = (Almost- no- response) and occurrence= (almost- no- connect) then decision = weak- reject. Facts: read/ write- ack- rate= 0.1, occurrence= Almost no- connect.

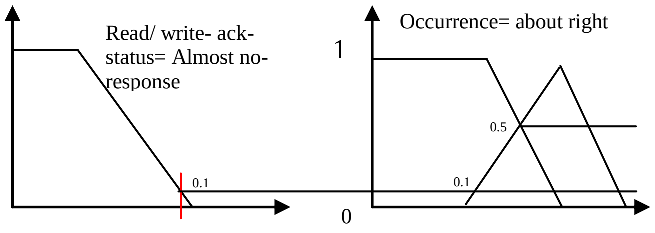


Fig (3-b): Rule: If read/ write- Ack- status = Almost no- response and occurrence= about right then decision= weak- reject. Facts: read/ write- Ack- rate= 0.1 occurrence= Almost no- connect.

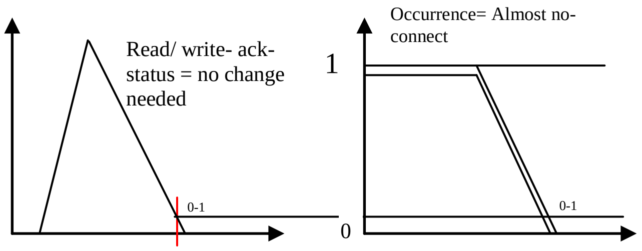

Read/ write- ack- status = no change needed
Occurrence= Almost no- connect
1
0-1
0-1
0

Fig (3-C): Rule: If read/ write- Ack- status = no change needed and occurrence= Almost no- connect then decision= weak- reject facts: read/ write- ack- rate= 0.1, (occurrence= Almost no- connect).

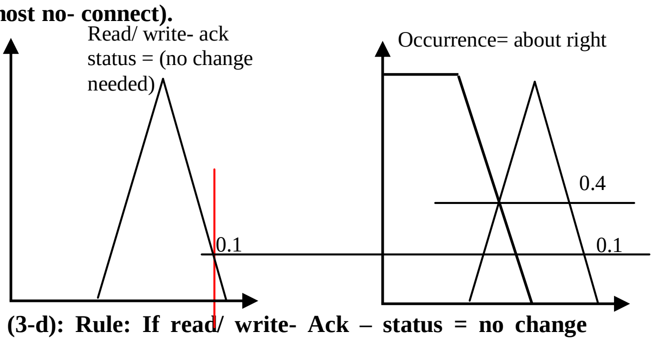


Fig (3-d): Rule: If read/ write- Ack – status = no change needed and occurrence= about right then decision= weak – reject. Facts: read/ write- Ack- rate= 0.1, occurrence= almost no- connect.

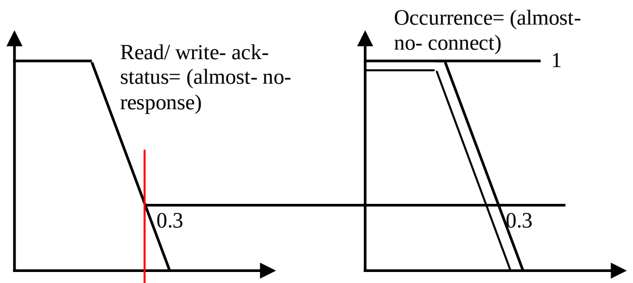


Fig (4-a): Rule: If read/ write- Ack – status = (Almost no-response) and occurrence= Almost no- connect then decision= weak – reject Facts: read/ write- Ack- rate= 0.3, occurrence= almost no- connect

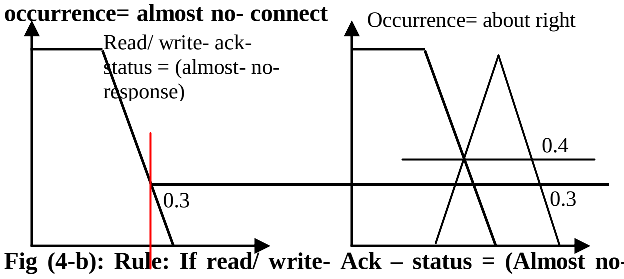


Fig (4-b): Rule: If read/ write- Ack – status = (Almost no-response) and occurrence= about right then decision= weak – select. Facts: read/ write- Ack- rate= 0.3, occurrence= almost no- connect

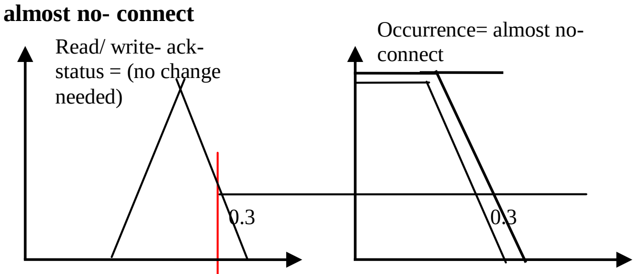


Fig (4-c): If read/ write- Ack – status= no change needed and occurrence= Almost no- connect then decision= weak–reject. Facts: read/ write- Ack- rate= 0.3, occurrence= almost no-connect

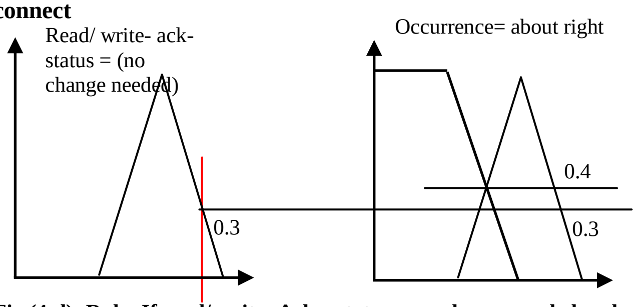


Fig (4-d): Rule: If read/ write- Ack – status = no change needed and occurrence= about right then decision= strong- select .Facts: read/ write- Ack- rate= 0.3, occurrence= almost no- connect

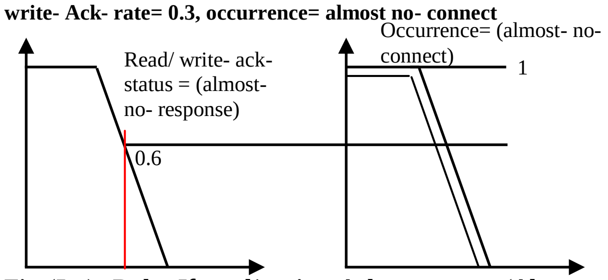


Fig (5-a): Rule: If read/ write- Ack – status = (Almost no-response) and occurrence= Almost no- connect then decision= weak – reject Facts: read/ write- Ack- rate= 0.6, occurrence= almost no- connect.

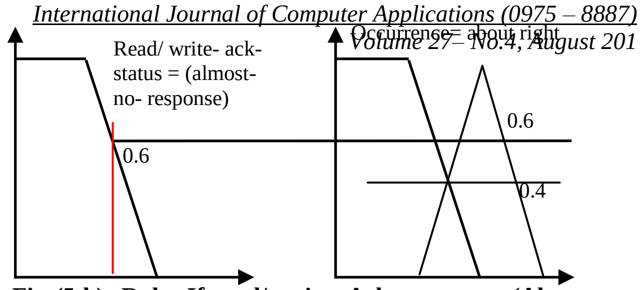


Fig (5-b): Rule: If read/ write- Ack – status = (Almost no-response) and occurrence= about right then decision= weak – reject. Facts: read/ write- Ack- rate= 0.6, occurrence= almost no- connect

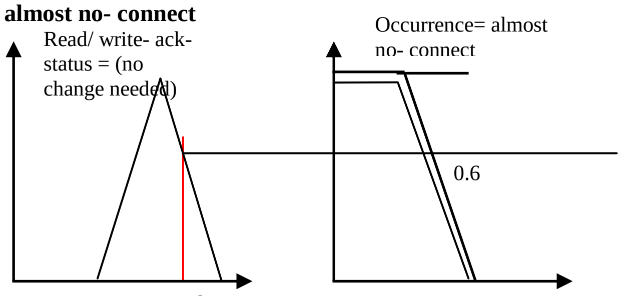


Fig (5-c): Rule: If read/ write- Ack – status = no change needed and occurrence= Almost no- connect then decision= weak – reject facts: read/ write- Ack- rate= 0.6, occurrence= almost no- connect

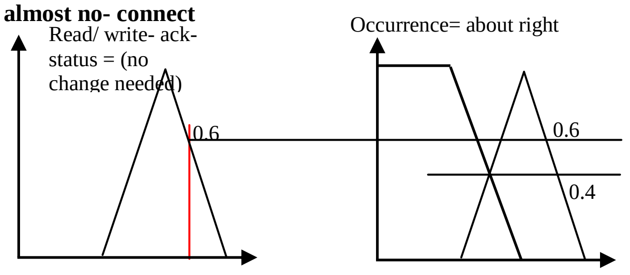


Fig (5-d): Rule: If read/ write- Ack – status = no change needed and occurrence= about right then decision= weak – select. Facts: read/ write- Ack- rate= 0.6, occurrence= almost no- connect

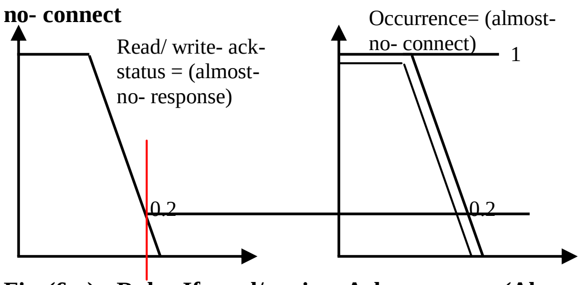


Fig (6-a): Rule: If read/ write- Ack – status= (Almost no-response) and occurrence= almost no- connect then decision= weak – reject Facts: read/ write- Ack- rate= 0.2, occurrence= almost no- connect

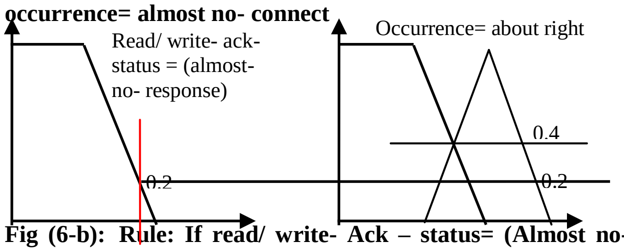


Fig (6-b): Rule: If read/ write- Ack – status= (Almost no-response) and occurrence= about right then decision= weak – select. Facts: read/ write- Ack- rate= 0.2, occurrence= almost no- connect

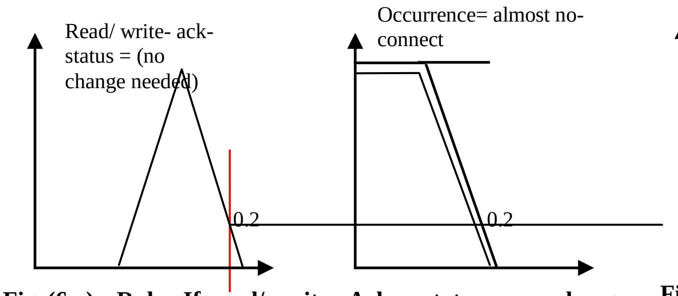

Read/ write- ack- status = (no change needed)
Occurrence= almost no- connect
0.2
0.2

Fig (6-c): Rule: If read/ write- Ack – status = no change needed and occurrence= Almost no- connect then decision= weak – reject Facts: read/ write- Ack- rate= 0.2, occurrence= almost no- connect

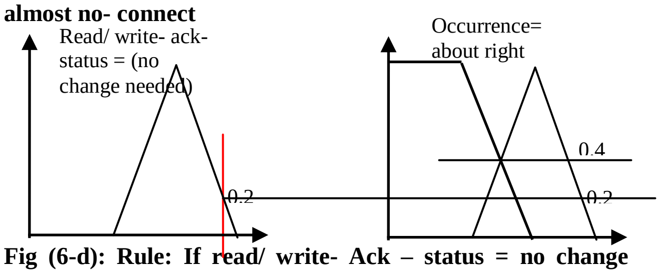


Fig (6-d): Rule: If read/ write- Ack – status = no change needed and occurrence= about right then decision= weak – select. Facts: read/ write- Ack- rate= 0.2, occurrence= almost No connect

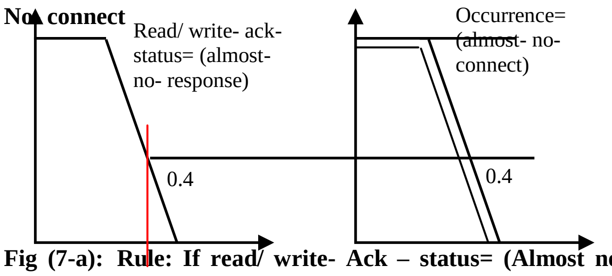

Read/ write- ack- status= (almost- no- response)
Occurrence= (almost- no- connect)
0.4
0.4

Fig (7-a): Rule: If read/ write- Ack – status= (Almost no-response) and occurrence= Almost no- connect then decision= weak – reject facts: read/ write- Ack- rate= 0.4, occurrence= almost no- connect

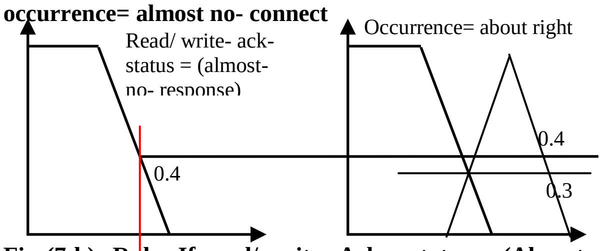


Fig (7-b): Rule: If read/ write- Ack – status = (Almost no-response) and occurrence= about right then decision= weak - reject. Facts: read/ write- Ack- rate= 0.4, occurrence= almost no- connect

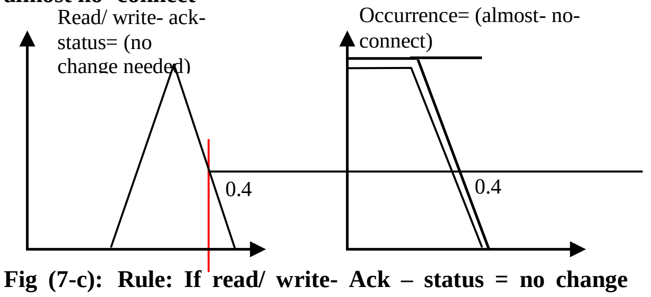

Read/ write- ack- status= (no change needed)
Occurrence= (almost- no- connect)
0.4
0.4

Fig (7-c): Rule: If read/ write- Ack – status = no change needed and occurrence= Almost no- connect then decision= weak – reject Facts: read/ write- Ack- rate= 0.4, occurrence= almost no- connect

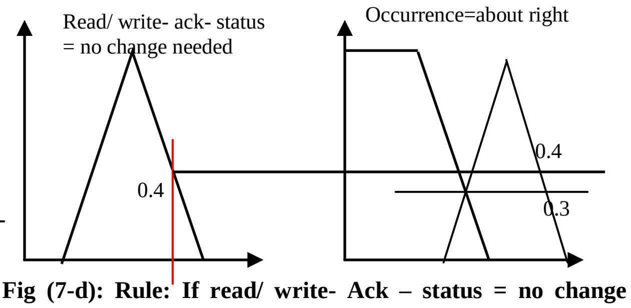


Fig (7-d): Rule: If read/ write- Ack – status = no change needed and occurrence about right then decision= strong-select. Facts: read/ write- Ack- rate= 0.4, occurrence= almost no- connect

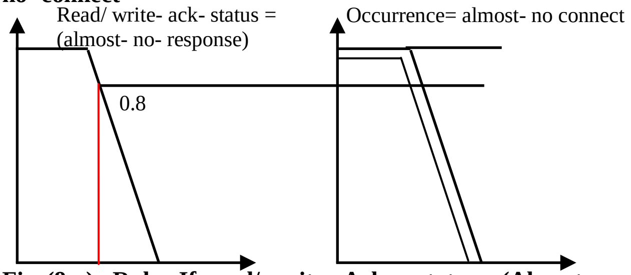

Read/ write- ack- status = (almost- no- response)
Occurrence= almost- no connect
0.8

Fig (8-a): Rule: If read/ write- Ack – status= (Almost no-response) and occurrence= Almost no- connect then decision= weak – select Facts: read/ write- Ack- rate= 0.8, occurrence= almost no- connect

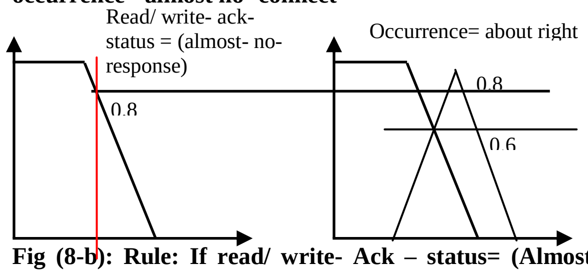


Fig (8-b): Rule: If read/ write- Ack – status= (Almost no-response) and occurrence= about right then decision= weak – select. Facts: read/ write- Ack- rate= 0.8, occurrence= almost no- connect

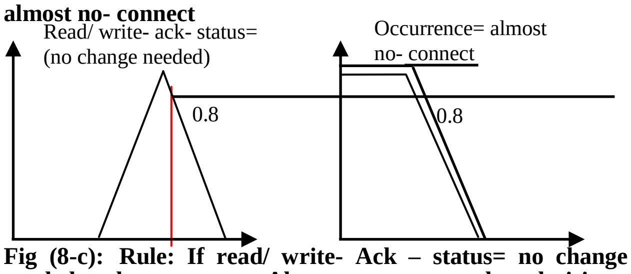

Read/ write- ack- status= (no change needed)
Occurrence= almost no- connect
0.8
0.8

Fig (8-c): Rule: If read/ write- Ack – status= no change needed and occurrence= Almost no- connect then decision= weak – select Facts: read/ write- Ack- rate= 0.8, occurrence= almost no- connect

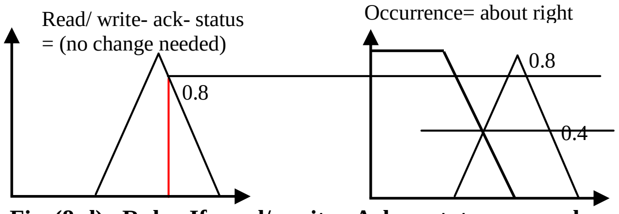


Fig (8-d): Rule: If read/ write- Ack – status = no change needed and occurrence= about right then decision= weak – reject. Facts: read/ write- Ack- rate= 0.8, occurrence= almost no- connect

4. PERFORMANCE EVALUATION FOR IMPLEMENTING ATOMIC READ/ WRITE SHARED MEMORY IN MOBILE AD HOC NETWORK USING FUZZY LOGIC.

Let us consider these assumptions as follow:
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.9).

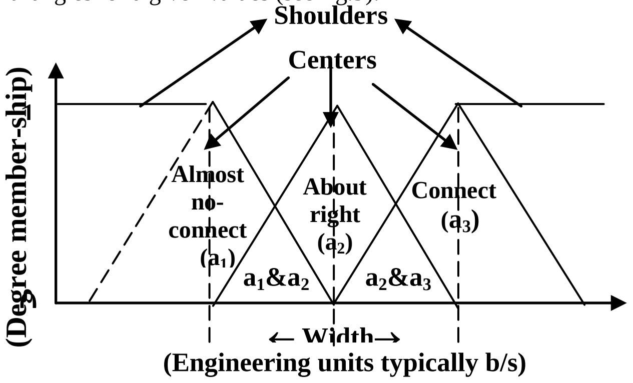
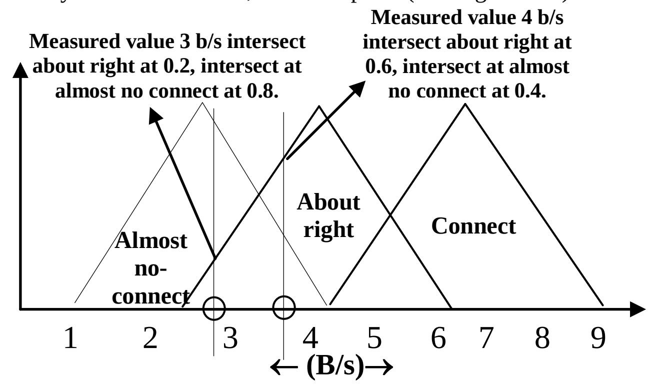


Fig.9 Membership Functions

5- Associating the above inputs and outputs as causes and effects with rules charts, as in the next fig.10 below (the cause – effect), the chart is made of triangles, the use of which will be explained. Triangles work just fine and are easy to work with width of the triangles can vary [14] [15]. 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).


Measured value 3 b/s intersect about right at 0.2, intersect at almost no connect at 0.8.
Measured value 4 b/s intersect about right at 0.6, intersect at almost no connect at 0.4.
Almost no- connect
About right
Connect
1
2
3
4
5
6
7
8
9
← (B/s)→

6- Drawing "effect" (output determining) triangles with their value (h=3 b/s or 4 b/s) 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 11- (a) (b)). This "effect" triangle will be used to determine the controller output. The result is affected by the width we have given the triangles and will be calculated. Note that the no change need state has a height of 0.2, 0.6 and the Almost no- response state has a height of 0.8, 0.4 because these were the intersect points for their matching "cause" triangles.

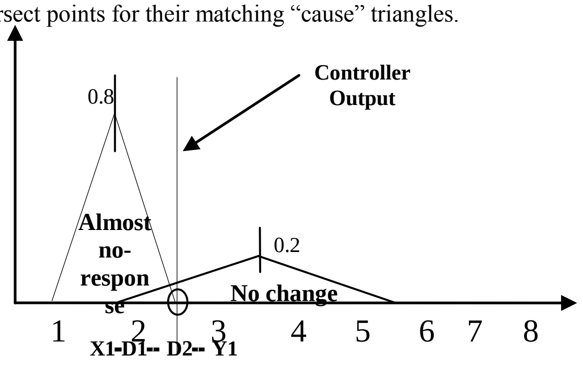


Fig. (11-a): Determination of Controller Output.

The output as seen in fig (11-a) is determined by calculating the point at which balance the two triangles, as follow:
The area of no change need triangle is ½ × 0.2 × 5=0.5
The area of Almost No- response triangle is ½ × 0.8 × 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: $D_1 + D_2 = 2.5 \rightarrow D_1 = 2.5\text{-}D_2$
So, by substituting (2-5- D_2) for D_1 in equation1 gives D_2
$0.8\times (2.5\text{- } D_2) = 0.5\ D_2$
$2\text{-}0.8\ D_2 = 0.5\ D_2$
$2=1.3\ D_2 \rightarrow D_{2=}\ 1.5$
$\therefore D_1 =1$

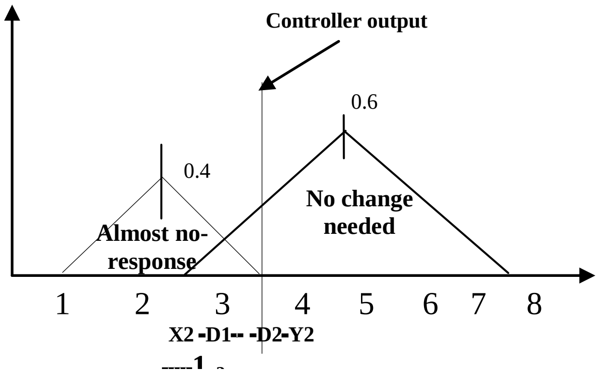


Fig (11-b): Determination of Controller Output.

The output as seen in fig (11-b) is determined by calculating the point at which balance the two triangles, as follow:
The area of no change need triangle is ½ × 6 × 0.6= 1.8
The area of Almost No- response triangle is ½ × 3 × 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

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

$2.1 - 0.6\, D_2 = 1.8\, D_2$

$2.1 = 1.8\, D_2 + 0.6\, D_2$

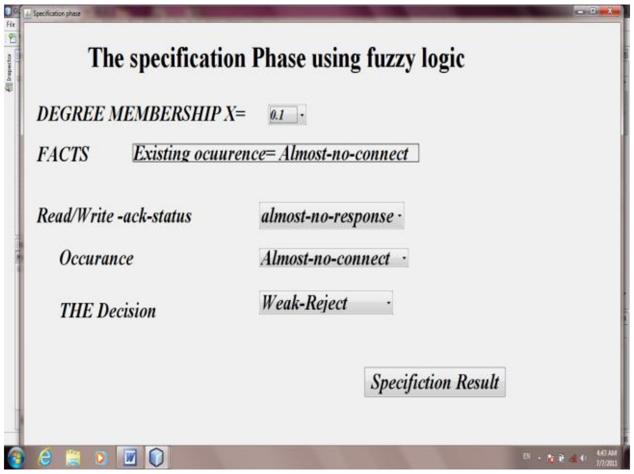
$2.1 = 2.4\, D_2 \rightarrow D_2 \approx 0.9$

$\therefore D_1 = 2.6$

Note, we are only discussing samples at instant values with a resulting controller output; the controller is sampling several times each second with a resulting "correction" output following each sample.

5. JAVA CODE FOR SPECIFICATION PHASE OF THE GEOQUORUM APPROACH

The java code for Specification Phase of the Geoquorum Approach for Implementing Atomic Read /Write Shared Memory in Mobile Ad Hoc Networks is considered GUI which determines specification phase code, this code will be illustrated in fig.12 and see(Appendix-A).


The specification Phase using fuzzy logic

DEGREE MEMBERSHIP X= 0.1

FACTS Existing ocuurence= Almost-no-connect

Read/Write -ack-status almost-no-response

Occurance Almost-no-connect

THE Decision Weak-Reject

Fig.12 Specification phase of the Geoquorum Approach in Java

6. CONCLUSIONS

First, in this paper the results of the previous figures which are done by fuzzy logic can be concluded as follow : the status of read/write operation in our application is always (almost no response or no change needed of the connection in the network), the future occurrence of the connection is always (almost no connect or about right of the connection in the network).Also, in these figures we have always two facts that the current status of occurrence is almost no connect and according to the fuzzy logic we assume X values which are determined at the range from 0 to 1. So the results of the connection by the network are either weak reject or weak select decision. Finally, at X is nearly equals 0.5 and the read/write Ack status is no change needed and the future occurrence of the connection is about right the result of the connection by the network is may be strong select decision. Second, we construct a java code for the specification phase of the geoquorum approach which is considered a communication protocol based on asynchronous real time distributed systems. This code is a tool to guarantee the accuracy and the validation of some phases of software lifecycle of this application as specification phase. This code is illustrated and building by GUI in java language.

7. REFERENCES

[1] Felix Bachman, Len Bass, C Buhman, S Comella-Dorda, F Long, J Robert, R Seacord, Kurt Wallnau," Technical concepts of component-based software engineering," Technical Report CMU/SEI-2000-TR-008, Carneggie 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] Y.Daniel Liang: Introduction to Java Programming. 6th Edition. USA: Pearson Prentice Hall, 2004.

[4] C.Snook, R.Harrison," Experimental Comparison of the Compressibility of a Z Specification and Its Implementation in Java", In: Proceeding Of Information and Software Technology 46(2004), PP: 955-971.

[5] 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).

[6] Haas, Z.J., Liang, and B.A, D.Wjghs. "Ad Hoc Mobile Management with Uniform GeoQuorums Systems", In: Proceeding of IEEE/ACM Transactions on Mobile ad hoc Networks 7(2), PP: 228-240(2004).

[7] 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

[8] 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

[9] 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

[10] 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 2005

[11]Ioana Şora, D. Todinca," Specification-based Retrieval of Software Components through Fuzzy Inference", in Acta Polytechnica Hungarica. Vol. 3, No. 3, 2006..

[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, 2007, 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, 2009, pp. 1–24.

[14] T. Lin, "Mobile Ad-hoc Network Routing Protocols: Methodologies and Applications," PhD thesis, Virginia Polytechnic Institute and State University, 2004.

[15] V. D. Tracy Camp, Jeff Boleng, "A Survey Of Mobility Models For Ad Hoc Network Research," Wireless Communications and Mobile Computing, 2:483–502, 2002.

8. APPENDIX-A: JAVA CODE

```
package examples;
import javax.swing.JOptionPane;
public class specification extends javax.swing.JFrame {


/** Creates new form specification */
public specification ( ) {
initComponents( );
    }
private void initComponents( ) {
jLabel1 = new javax.swing.JLabel( );
jLabel2 = new javax.swing.JLabel( );
jLabel3 = new javax.swing.JLabel( );
jComboBox1 = new javax.swing.JComboBox( );
jLabel4 = new javax.swing.JLabel( );
jComboBox2 = new javax.swing.JComboBox( );
jLabel5 = new javax.swing.JLabel( );
jComboBox3 = new javax.swing.JComboBox( );
jLabel6 = new javax.swing.JLabel( );
jComboBox4 = new javax.swing.JComboBox( );
jLabel7 = new javax.swing.JLabel( );
jButton1 = new javax.swing.JButton( );
setDefaultCloseOperation(javax.swing.Window
Constants.DISPOSE_ON_CLOSE);
setTitle("Specification phase\n");
jLabel1.setFont(new java.awt.Font("Times
New Roman", 1, 14));
jLabel1.setText("FACTS");
jLabel2.setFont(new java.awt.Font("Tahoma",
1, 18));
jLabel2.setForeground(new
java.awt.Color(102, 0, 102));
jLabel2.setText("The specification Phase
using fuzzy logic");
jLabel3.setFont(new    java.awt.Font("Times
New Roman", 1, 14));
jLabel3.setText("READ/WRITE -ACK-STATUS");
jComboBox1.setFont(new
java.awt.Font("Tahoma", 1, 14));
jComboBox1.setModel(newjavax.swing.DefaultC
omboBoxModel(newString[]{"almost
noresponse","no-change-needed", " " }));
jComboBox1.addActionListener(new
java.awt.event.ActionListener( ) {
public void actionPerformed(   java. awt.
event. ActionEvent evt) {
jComboBox1ActionPerformed(evt);
            }
        });
jLabel4.setFont(new java.awt.Font("Tahoma",
1, 14));
jLabel4.setText("DEGREE MEMBERSHIP X=");
jComboBox2.setFont(new
java.awt.Font("Tahoma", 1, 14));
jComboBox2.setModel(new
javax.swing.DefaultComboBoxModel(new
String[] {"0.1", "0.3", "0.2","0.4", "0.8",
"0.6" }));
jComboBox2.addActionListener(new
java.awt.event.ActionListener() {
public void actionPerformed(   java. awt.
event. ActionEvent evt) {
jComboBox2ActionPerformed(evt);
            }
        });
jLabel5.setFont(new java.awt.Font("Tahoma",
1, 14));
Label5.setText("OCCURANCE");jComboBox3.setF
ont(new java.awt.Font("Times New Roman", 1,
14));
```

```
jComboBox3.setModel(newjavax.swing.DefaultC
omboBoxModel(newString[]{"Almost-no-

connect"," About-right" }));

jComboBox3.addActionListener(new
java.awt.event.ActionListener( ) {

public void actionPerformed(   java.  awt.
event. ActionEvent evt)

  {

jComboBox3ActionPerformed(evt);

            }

        });

jLabel6.setFont(new java.awt.Font("Tahoma",
1, 14));

jLabel6.setText("THE DECISION");

jComboBox4.setFont(new
java.awt.Font("Tahoma", 1, 14));

jComboBox4.setModel(new
javax.swing.DefaultComboBoxModel(new
String[] {"WEAK-REJECT",

"STRONG-SELECT", "WEAK-SELECT" }));

jLabel7.setFont(new
java.awt.Font("Verdana", 1, 12));

jLabel7.setForeground(new
java.awt.Color(102, 0, 102));

jLabel7.setText("    Existing    occurrence=
Almost-no-connect");

jLabel7.setBorder(javax.swing.BorderFactory
.createLineBorder(new java.awt.Color(0,  0,
0)));

jButton1.setText("Specification Result");

jButton1.addActionListener(new
java.awt.event.ActionListener( ) {

public  void  actionPerformed  (java.  awt.
event. ActionEvent evt) {

jButton1ActionPerformed (evt);

            }

        });

javax.swing.GroupLayout   layout   =   new
javax.swing.GroupLayout(getContentPanel());

getContentPanel().setLayout(layout);

layout.setHorizontalGroup(layout.createPara
llelGroup(javax.swing.GroupLayout.Alignment
.LEADING)

.addGroup(layout.createSequentialGroup( )

.addGroup(layout.createParallelGroup(javax.
swing.GroupLayout.Alignment.LEADING).addGro
up(layout.createSequentialGroup(
).addGap(140, 140, 140)

.addComponent(jLabel2,javax.swing.GroupLayo
ut.PREFERRED_SIZE,457,javax.swing.GroupLayo
ut.PREFERRED_SIZE))
.addGroup(layout.createSequentialGroup( )

.addGroup(layout.createParallelGroup(javax.
swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup(
).addGap(28, 28, 28)

.addGroup(layout.createParallelGroup(javax.
swing.GroupLayout.Alignment.LEADING)
.addComponent(jLabel4).addComponent(jLabel1
,javax.swing.GroupLayout.PREFERRED_SIZE,60,
javax.swing.GroupLayout.PREFERRED_SIZE)))
.addGroup(layout.createSequentialGroup(    )
.addGap(25,25,25)
.addGroup(layout.createParallelGroup(javax.
swing.GroupLayout.Alignment.LEADING)
.addComponent(jLabel3)

.addComponent(jLabel6,javax.swing.GroupLayo
ut.PREFERRED_SIZE,126,javax.swing.GroupLayo
ut.PREFERRED_SIZE).addComponent(jLabel5,jav
ax.swing.GroupLayout.PREFERRED_SIZE,114,
javax.swing.GroupLayout.PREFERRED_SIZE))
.addGap(4, 4, 4)))

.addPreferredGap(javax.swing.LayoutStyle.Co
mponentPlacement.RELATED)
.addGroup(layout.createParallelGroup(javax.
swing.GroupLayout.Alignment.LEADING)

.addComponent(jComboBox1,javax.swing.GroupL
ayout.PREFERRED_SIZE,224,javax.swing.GroupL
ayout.PREFERRED_SIZE).addComponent(jComboBo
x3, javax.swing.GroupLayout.PREFERRED_SIZE,
181,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addComponent(jComboBox4,javax.swing.GroupL
ayout.PREFERRED_SIZE,220,javax.swing.GroupL
ayout.PREFERRED_SIZE)

.addComponent(jComboBox2,javax.swing.GroupL
ayout.PREFERRED_SIZE,56,javax.swing.GroupLa
yout.PREFERRED_SIZE).addComponent(jLabel7,j
avax.swing.GroupLayout.PREFERRED_SIZE,277,j
avax.swing.GroupLayout.PREFERRED_SIZE)).add
Gap(44,44,44)).addGroup(layout.createSequen
tialGroup(    ).addGap(238,    238,    238)
.addComponent(jButton1))).addContainerGap(4
21, Short.MAX_VALUE))

        );

layout.setVerticalGroup(layout.createParall
elGroup(javax.swing.GroupLayout.Alignment.L
EADING).addGroup(layout.createSequentialGro
up().addGap(29,29,29).addComponent(jLabel2)
.addGap(50, 50, 50)

.addGroup(layout.createParallelGroup(javax.
swing.GroupLayout.Alignment.BASELINE)

.addComponent(jLabel4,javax.swing.GroupLayo
ut.PREFERRED_SIZE,30,
javax.swing.GroupLayout.PREFERRED_SIZE).add
Component(jComboBox2,

javax.swing.GroupLayout.PREFERRED_SIZE,java

x.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addGap(32, 32, 32)

.addGroup(layout.createParallelGroup(javax.
swing.GroupLayout.Alignment.TRAILING,
false)

.addGroup(layout.createSequentialGroup( )

.addComponent(jLabel1,javax.swing.GroupLayo
ut.PREFERRED_SIZE,26,javax.swing.GroupLayo
ut.PREFERRED_SIZE).addGap(114,114,114)).add
Group(layout.createSequentialGroup()
.addGap(4, 4, 4)

.addComponent(jLabel7,javax.swing.GroupLayo
ut.PREFERRED_SIZE,30,javax.swing.GroupLayou
t.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.Co
mponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE,Short.
MAX_VALUE).addGroup(layout.createParallelGr
oup(javax.swing.GroupLayout.Alignment.BASEL
INE).addComponent(jComboBox1,
javax.swing.GroupLayout.PREFERRED_SIZE,25,j
avax.swing.GroupLayout.PREFERRED_SIZE).addC
omponent(jLabel3,javax.swing.GroupLayout.PR
EFERRED_SIZE27,javax.swing.GroupLayout.PREF
ERRED_SIZE)).addGap(43, 43, 43)))

.addPreferredGap(javax.swing.LayoutStyle.Co
mponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.
swing.GroupLayout.Alignment.LEADING)

.addComponent(jComboBox3,javax.swing.GroupL
ayout.PREFERRED_SIZE,27,javax.swing.GroupLa
yout.PREFERRED_SIZE)
.addComponent(jLabel5)).addGap(26, 26, 26)

.addGroup(layout.createParallelGroup(javax.
swing.GroupLayout.Alignment.BASELINE)

.addComponent(jComboBox4,javax.swing.GroupL
ayout.PREFERRED_SIZE,javax.swing.GroupLayou
t.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addComponent (jLabel6)).addGap (30, 30, 30

.addComponent (jButton1)

.addContainerGap (1164, Short.MAX_VALUE))

        );

pack ( );

    }// </editor-fold>

PrivatevoidjButton1ActionPerformed
(java.awt.event.ActionEvent evt) {

JOptionPane.showMessageDialog (null,

"The specification phases completed
successfully");

    }

privatevoidjComboBox1ActionPerformed(java.a
wt.event.ActionEvent evt) {

    }

privatevoidjComboBox2ActionPerformed(java.a
wt.event.ActionEvent evt) {

    }

privatevoidjComboBox3ActionPerformed(java.a
wt.event.ActionEvent evt) {

// TODO add your handling code here:

    }


public static void main (String args [])
{java.awt.EventQueue.invokeLater(new
Runnable ( ) {

public void run ({new specification (
).setVisible(true);

            }

        });

      }

// Variables declaration - do not modify

private javax.swing.JButton jButton1;

private javax.swing.JComboBox jComboBox1;

private javax.swing.JComboBox jComboBox2;

private javax.swing.JComboBox jComboBox3;

private javax.swing.JComboBox jComboBox4;

private javax.swing.JLabel jLabel1;

private javax.swing.JLabel jLabel2;

private javax.swing.JLabel jLabel3;

private javax.swing.JLabel jLabel4;

private javax.swing.JLabel jLabel5;

private javax.swing.JLabel jLabel6;

private javax.swing.JLabel jLabel7;

    // End of variables declaration
```