The Comprehensibility of a Z Specification Language and Its Implementation in JAVA for Implementing Atomic Read/Write Shared Memory in Mobile Ad hoc Network

Reham.A.Shihata
The Comprehensibility of a Z Specification Language and Its Implementation in JAVA for Implementing Atomic Read/Write Shared Memory in Mobile Ad hoc Network

Reham.A. Shihata
PhD in Computer Science, EL Menoufia University – Egypt. Software Consultant in Egyptian Syndicate of Programmers & Scientists
anwerreham@yahoo.com

Abstract
Comprehensibility is often raised as a problem with formal notations; yet formal methods practitioners dispute this. In a survey, one interview said "formal specifications are no more difficult to understand than code". Measurement of comprehension is necessarily. In this paper, a comprehension of Z specification with that of its implementation in JAVA for atomic object Read/Write shared memory in mobile ad hoc network is performed.

Keywords: Z Specification Language, Formal Specification, GUI, Java Language, Mobile Ad Hoc Network.
1. Introduction

The Geoquorums approach for implementing atomic read/write shared memory in mobile ad hoc networks, this approach is based on associating abstract atomic objects with certain geographical locations this algorithm for mobile ad hoc networks which uses no pre-existing infrastructure unlike cellular networks that depend on fixed, wired base stations [1] [2][3]. Instead the network is formed by the mobile nodes themselves, which co-operate to route communication from sources to destinations.

Specify an Atomic in Geoquorums Approach Object as a Variable Type[4][5][6]:

\[ V_1 \] a set of legal values (i.e states) for the object

\[ v_0 \in V_1 \] an initial value (i.e states) for the object

Invocations, a set of invocations

Responses , a set of responses

\[ \delta \] the transition \( F_\delta \) a mapping from: \( \text{(invocations} \times V) \) \( \xrightarrow{\delta} \) \( \text{(responses} \times V) \)

Mathematical Notation for Geoquorums Approach:

- \( I \) the totally-ordered set of node identifiers.

- \( i_0 \in I \), a distinguished node identifier in \( I \) that is smaller than all order identifiers in \( I \).

- \( S \), the set of port identifiers, defined as \( N^{\geq 0} \times \text{OP} \times I \),

  where \( \text{OP} = \) (get, put, confirm, recon-done).

- \( O \), the totally-ordered, finite set of focal point identifiers.

- \( T \), the set of tags defined as \( R^{\geq 0} \times I \).

- \( U \), the set of operation identifiers, defined as \( R^{\geq 0} \times S \).

- \( X \), the set of memory locations for each \( x \in X \).

  - \( V_x \) the set of values for \( x \)

  - \( v_{0.x} \in V_x \), the initial value of \( X \)

- \( M \), a totally-ordered set of configuration names

  - \( c_0 \in M \), a distinguished configuration in \( M \) that is smaller than all other names in \( M \).

- \( C \), totally-ordered set of configuration identifies, as defined as:

  \( R^{\geq 0} \times I \times M \)

- \( L \), set of locations in the plane, defined as \( R \times R \)

An Atomic Object is Specified as a Variable type, \( \tau \), consists of
V, a set of legal values (i.e states) for the object

- \( v_0 \in V \) an initial value (i.e states) for the object.
- invocations, a set of invocations.
- responses, a set of responses
- \( \delta \), the transition \( F_\delta \), a mapping from:
  \[(\text{invocations} \times V) \rightarrow (\text{responses} \times V)\]

That maps every invocation and state to a response and a new state.

**Specify a Read/Write object as a Variable Type [7][8][9]**

\( V \), as arbitrary set of values for the atomic object

\( v_0 \in V \) an arbitrary initial value

invocations= \( \{ \text{read} \} \cup \{ \text{write} \colon v \in V \} \)

responses= \( \{ \text{read-ack} \colon v \in V \} \cup \{ \text{write-ack} \} \)

\( \delta \) is defined as:

- \( \delta \colon (\text{read} , v) \rightarrow \langle \text{read-ack} (v), v \rangle \)
- \( \delta \colon (\text{write} , v) \rightarrow \langle \text{write-ack} , v \rangle \)

**Formal Sequential Specification for Abstract Read/Write Object[10][11][12]:**

State

Value, initially \( v_0 \)

Operation

Read ( )

Return read-ack (value)

Write (new-value)

Value \( \leftarrow \) new-value

return write-ack ( )

Canonical Atomic Object Specification of

\[ t = \langle V, v_0, \text{invocations}, \text{responses}, \delta \rangle, \text{ for the set } \varphi \text{ of ports} \]

Signature:

Input:
invoke \((\text{inv})_\rho\), \(\text{inv} \in \text{invocations}, \ p \in \varphi\), the invocations defined by the variable type \(\tau\)

Outputs:

Respond \((\text{resp})_\rho\), \(\text{resp} \in \varphi\), the responses defined by the variable type \(\tau\)

Internal:

Perform \((\text{inv}, v, \text{resp}, v')_\rho, \text{inv} \in \text{invocations}, \text{resp} \in \text{responses}, v,v' \in V, \ p \in \varphi\),

perform the transitions defined by the variable type \(\tau\)

State:

\(\text{val} \in V\) a value, \(\text{initially} V_0\)

\(\text{inv}-\text{buffer}, \text{a set of pairs,} \langle \text{inv, p} \rangle\) for invocations, \(\text{inv} \in \text{invocations}, \ p \in \varphi, \text{initially} \Phi\)

\(\text{resp}-\text{buffer}, \text{a set of pairs,} \langle \text{resp, p} \rangle\) for responses, \(\text{resp} \in \text{responses}, \ p \in \varphi, \text{initially} \Phi\)

Trasitions:

Input \(\text{invoke (inv)}_\rho\)

Effect:

\(\text{inv-buffer} \leftarrow \text{inv-buffer} \cup \{ \langle \text{inv, p} \rangle \}\)

Output respond \((\text{resp})_\rho\)

Precondition:

\(\langle \text{resp, p} \rangle \in \text{resp-buffer}\)

Effect:

\(\text{resp-buffer} \leftarrow \text{resp-buffer} \setminus \{ \langle \text{resp, p} \rangle \}\)

**Definition of The Put/Get Variable Type \(\tau\) [13][14][15]:**

put/get variable type \(\tau\)

State

\(\text{tag} \in T, \text{initially,} <0, i_o>\)

\(\text{value} \in V, \text{initially} V_0\)

\(\text{config-id} \in C, \text{initially} <0, i_o, C_0>\)

\(\text{Confirmed- set} \ T, \text{initially} \Phi\)

\(\text{recon ip, a Boolean, initially false}\)

Operations:-

Put (new-tag, new-value, new-config-id)
If (new-tag > tag) then
value ← new-value
tag ← new-tag
If (new - config-id > config-id) then
config-id ← new- config-id
recon-ip ← true
return put-ack (config-id, recon-ip)
get (new-config-id)
If (new-config-id > config-id) then
config-id ← new- config-id
Recon-ip ← true
Confirmed ← (tag ∈ confirmed-set)
return get-ack (tag, value, confirmed, config-id, recon-ip)
confirm (new-tag)
confirmed-set ← confirmed-set ∪ {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. Z Specification for Implementing Atomic Read/Write Shared Memory in Mobile Ad hoc Networks

**Objects**
Focal point object
type Fpo: atomic
object Fpo value: N
Op-recon-ip:
True/False Op. record:
record type Op. phase:
phase-Type Op. tag:
Variable
Op. value: Variable
**Signature**

**Type**
Signature: External type
Signature: Internal type

**Operation Manager Client**

**Type** OMC: put variable type
OMC: get variable type

**Focal Point Emulator Type**

< current- port-number, op, i > > 0

**Focal Point Emulator Client Type**

invoke/ respond interface: interface type < current -port-number, op, i > > 0

**Focal Point Emulator Server**

**Type** Local broadcast > 0
LBcast Service: Service type **Initialization**
put ?: IP variable type
get- ack- response: IP variable type get ?: IP variable type
put- ack- response: IP variable type tag ?: N order on values
confirm- ack- response: IP variable type config – id: parameter
recon- done- ack : IP variable type
    confirmed- set?:
    set ↔ tags recon- ip
    flag?: True/False
recon- done:
True/False

**Operations**

**Put- invocation**

∃ New- tag .int -type > tag. int-type
∃ New-Config- id. Parameter- type > config-id. Parameter-type
∃ recon- done= True
put- invocation’ = put-ack- response

**Get- invocation**
∃ new-config-id> config-id
get-invocation'= get-ack-response

**Confirm**  
*invocation*  ∃ new-
tag > tag

∃ confirmed-set = confirm-set ∪ {new-tag? :N} confirm-invocation= confirm-ack-response

**Recon**  
--done  
*invocation*  ∃ recon-ip= True

∃ new-config-id. parameter = config-id.
Parameter recon-done-invocation'= recon-done-ack

√ ~ conf-id=op.recon-conf-id→ recon-ip=false  **Success**

No - Error!: Success
No - Error!: Recon-Ip
No - Error!: Tag
No - Error!: Config - Id
Put-Ack-Response= Okay
Get-Ack-Response= Okay
Confirm-ack-response= Okay
Recon-Done-Ack= Okay

((Put-Type, Get-Type) ^ Success)
((Confirm-Type, Recon-Done-Type) ^ Success)

3. **JAVA Code for Implementing Atomic Read/Write Shared Memory in Mobile Ad hoc Networks**

Import java. Lang. Exception;
Class invariant exception extends Exception {
Public invariant exception (int id) {surper (id);} 

Class put type 

int new-tag -
type int tag-
type 

int new- config-
id int config- id-
type 

Public put type (int: new- tag) throws invariant exception { if (recon- done== true) { 
Invariant exception id =new invariant exception ("invariant: new-tag must be > 
=1") 
put- invocation= put –ack- response 
system.out.Println (" put- ack- response= okay") ;} } 

Class gettype 
{ int config-
id 

int new- config –id 

Public gettype (int: new -config- id) throws invariantexception { if (new- config- id > config- id){ 

invariantexception id= new 
invariantexception (" invariant: new-
config- id must be > =1"); get - invocation= get- ack- response} 

throw id; }
get-invocation=get-ack-response
system.out.Println (" get- ack- response=
okay"); } }

Class confirmtype {
Int op.recon- conf-
id Int new- tag
Int new- config-
id Int confirmed-
set
Public confirmtype (int: confirmed- set) throws
invariantexception if (new- tag> && confirmed – set {new-
tag}) {Invariantexception id= new invariantexception
(" invariant: confirmed- set must be
>=1); throw id;}
confirm- invocation = confirm-ack-
response system .out. println ("confirm- ack
–response"); }

Class recon-done type {
int new- config-id parameter

Public recon-done type (int: new- config-id) throw
invariantexception if (new- config-id parameter== config-id-
parameter){ invariantexception id= new invariantexception
(" invariant: new- config- id. Parameter
>=1")} throw id;
recon- done invocation= recon- done-
ack }

Public boolean recon –ip as (confirmtype: conf- id
confirm) {boolean recon- ip = true;
boolean op. recon- conf- id = true;

if (conf- id !== op.recon- conf- id) recon- ip = false; return recon- ip; }
}

system. out. println ( "confirm- ack-response"); }

Class adhoctype{
boolean ammadh=
true; boolean puttype=
true; boolean gettype=
true;
boolean confirmtype= true;
boolean recon -donetype=
true; boolean success = true;

Public ammadh type (boolean success) throws invariantexception

if ((puttype || gettype) && success) || (( confirmtype || recon- done type &&
success )) Invariantexception id= new invariantexception

(" invariant: success must be true"); throw id;)
put-ack-response=
okay; get- ack-
response= okay;
confirm- ack – response =
okay; recon –done –ack=
okay; ammadh= okay

system. out. println (" put- ack- response= okay");

system. out. println (" get- ack- response= okay");

system. out. println (" confirm- ack- response=
okay");

system. out. println (" recon- done-
response= okay"); }
3. Constructing a Java Code for Implementing Atomic Read /Write Shared Memory in Mobile Ad Hoc Networks

This code will be divided into four phases by using GUI in Java

Fig.1 The GUI of Analysis Phase Using Java Code

Fig.2 The GUI of Specification Phase Using Java Code
Fig. 3 The GUI of Design Phase Java Code

Fig. 4 The GUI of Testing Phase Using Java Code
4. Java Code for the Geoquorum Approach

The final code in mobile ad hoc network is done as follow:

Fig.5 The GUI for The Communication Protocol Based on Asynchronous Real Time Distributed Systems in Java
package examples;
import java.awt.Frame;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JOptionPane;

/**
 * @Author Reham
 **/

class NewJFrame extends javax.swing.JFrame {

    /** Creates new form NewJFrame */
    public NewJFrame() {
        initComponents();
    }

    /** This method is called from within the constructor to 
     * initialize the form.
     
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    private void initComponents() {
        buttonGroup1 = new javax.swing.ButtonGroup();
        jLabel1 = new javax.swing.JLabel();
jToggleButton1 = jToggleButton2 = jToggleButton3 =
jToggleButton4 =
new javax.swing.JToggleButton();
new javax.swing.JToggleButton();
new javax.swing.JToggleButton();
new javax.swing.JToggleButton();

jButton2 = new javax.swing.JButton();

addComponent(jToggleButton1, javax.swing.GroupLayout.DEFAULT_SIZE, 300, Short.MAX_VALUE);
addComponent(jToggleButton2, javax.swing.GroupLayout.PREFERRED_SIZE, 319, javax.swing.GroupLayout.PREFERRED_SIZE);
addComponent(jToggleButton4).addGap(163, 163, 163);

addGap(33, 33, 33).


layout.setVerticalGroup(
);
	pack();

}// </editor-fold

private void jToggleButton3ActionPerformed(java.awt.event.ActionEvent evt) {
rehaml  r1=new rehaml();
//r1.setSize(900,900);
r1.setVisible(true);
}

private void jToggleButton1ActionPerformed(java.awt.event.ActionEvent evt) {
NewJFrame1  f1=new NewJFrame1();
//f1.setSize(900,900);
f1.setVisible(true);
}

private void jToggleButton4ActionPerformed(java.awt.event.ActionEvent evt) {
NewJFrame2  f2=new NewJFrame2();
f2.setSize(900,900);
private void jTableButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    this.dispose();
}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new NewJFrame().setVisible(true);
        }
    });
}

// Variables declaration - do not modify
private javax.swing.ButtonGroup buttonGroup1;
private javax.swing.JButton jButton2;
private javax.swing.JLabel jLabel1;
private javax.swing.JToggleButton jToggleButton1;
private javax.swing.JToggleButton jToggleButton2;
private javax.swing.JToggleButton jToggleButton3;
private javax.swing.JToggleButton jToggleButton4; // End of variables declaration
}

Conclusions
In this paper, Z specification as a formal specification language and a java code are constructed for 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 development lifecycle of this application such as analysis phase, specification phase, design and testing phases. All these phases are illustrated and building by java language.

References